TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: N2612 – Elektrotechnika a informatika

Studijní obor: 1802T007 – Informační technologie

Testování webových aplikací

Web applications testing

Diplomová práce

Autor: **Bc. Jan Kanajlo**

Vedoucí práce: Mgr. Jiří Vraný, Ph.D.

V Liberci 16. 5. 2012

PROHLÁŠENÍ

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum 16. 5. 2012

Podpis

Na tomto místě bych chtěl co nejupřímněji poděkovat panu doktoru Jiřímu Vranému za odborné vedení, pozitivní přístup, konstruktivní kritiku a za veškerý čas, který mi věnoval. Rád bych také poděkoval své rodině za podporu při studiu.

Jan Kanajlo

ABSTRAKT

Česky

Tato diplomová práce se zabývá vývojem webové aplikace pro simulaci činnosti deterministického konečného automatu. Teoretická část diplomové práce je zaměřena na problematiku testování webových aplikací. V diplomové práci jsou popsány jednotlivé druhy testování a nástroje, které tyto typy testů realizují. V praktické části je shrnut výběr technologií, návrh a implementace aplikace. Též je zde vyložen přínos jednotlivých testovacích nástrojů, které byly při vývoji aplikace použity. Výsledkem diplomové práce je funkční webová aplikace a také ucelený přehled testovacích přístupů, které lze užít při vývoji webových aplikací.

Klíčová slova: deterministický konečný automat, webová aplikace, testování

Anglicky

This thesis deals with the development of web application for simulation of deterministic finite automaton. The theoretical part is focused on web applications testing. This thesis describes different types of tests and tools for web applications testing. The practical section summarizes selection of technologies, design and implementation of application. Also there is the contribution of each test tool, that was used for development of application. The result of this thesis is the developed application and the comprehensive result is overview of testing approaches, that can be used in web development.

Keywords: deterministic finite automata, web application, testing

OBSAH

SEZNAM ZKRATEK

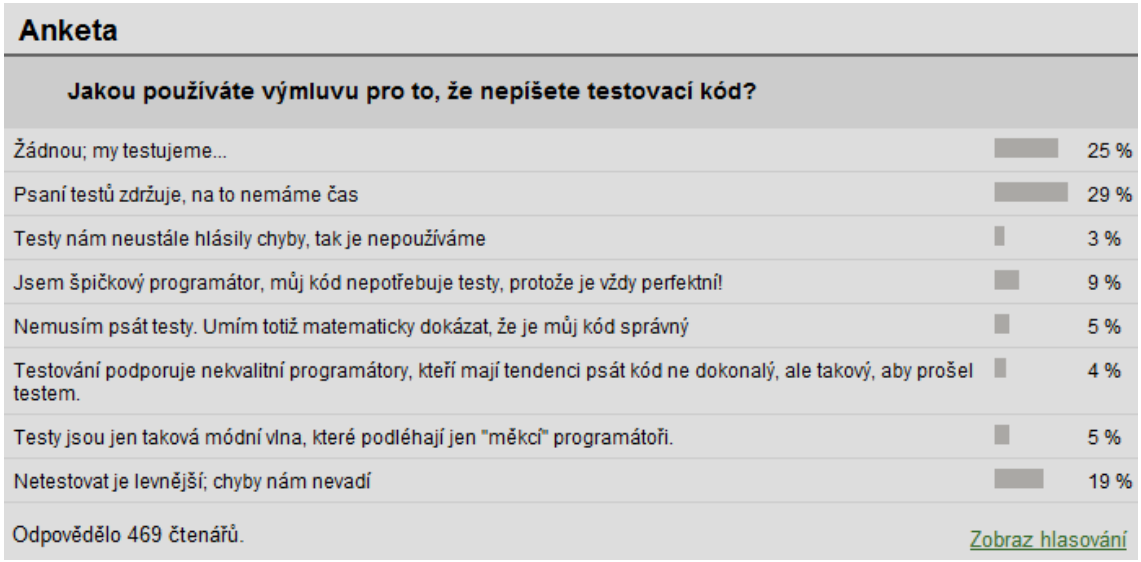
CSS	Cascading Style Sheets
DFA	Deterministic Finite Automata
FTP	File Transfer Protocol
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Enviroment
IMAP	Internet Message Access Protocol
kB	Kilobyte
LAMP	Linux, Apache, MySQL, PHP (Python, Perl)
MVC	Model View Controller
PHP	Hypertext Preprocessor
Png	Portable Network Graphics
POP3	Post Office Protocol 3
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TDD	Test Driven Development
URL	Uniform Recource Locator
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XSS	Cross-site scripting

SEZNAM OBRÁZKŮ

1 Úvod

Tato diplomová práce navazuje na autorův magisterský projekt, ve kterém byl vyvíjen simulátor deterministického konečného automatu v podobě webové aplikace. Simulátory deterministického konečného automatu sice existují, ovšem pouze v podobě desktopových aplikací. Vzhledem k budoucímu užití aplikace pro výukové účely byl kladen velký důraz na návrh jednoduchého a srozumitelného uživatelského rozhraní. Stěžejní částí aplikace je animace průchodu slova definovaným automatem, která by měla být interaktivní a pro uživatele srozumitelná. Diplomová práce se také zabývá návrhem další funkcionality aplikace, která by rozšířila možnosti použití aplikace a také usnadnila uživatelům použití. Aplikace byla vyvinuta na vývojové platformě LAMP, jakožto nejdostupnější a nejpoužívanější platformě pro vývoj webových aplikací.

Při vývoji aplikace bylo cílem dosáhnout co možná nejvyšší kvality aplikace ve smyslu použitelnosti, spolehlivosti a znovupoužitelnosti kódu. S ohledem na tento požadavek se teoretická část práce zabývá problematikou testování webových aplikací. Tato problematika se v současné době jeví jako velmi aktuální, neboť v praxi je zřejmý trend, kdy velké množství aplikací mění podobu z desktopové verze na verzi webovou. V důsledku tohoto trendu sílí požadavek na kvalitu webových aplikací, kterou by mělo zajistit právě efektivní testování a dá se předpokládat, že v blízké budoucnosti bude mít testování webových aplikací mnohem větší význam, než je tomu nyní. O podceňování procesu testování webových aplikací svědčí, kromě nedostatku odborných publikací na toto téma, také signály z praxe, kde jsou některé přístupy testování považovány programátory za zbytečné a nepřínosné. Tento přístup je způsoben především kladením krátkodobých cílů ve smyslu co nejlevnějšího a nejrychlejšího vývoje aplikace, ovšem z dlouhodobého hlediska se tento přístup zcela jistě negativně projeví na udržovatelnosti a rozšitelnosti aplikace. Jako názorná ukázka přístupu některých programátorů může posloužit anketa na obrázku 1.1, která je převzata z webu Zdroják [1], který se zabývá problematikou tvorby webových aplikací.



Obrázek 1.1 – Anketa ilustrující přístup programátorů k testování

V teoretické části diplomová práce je definováno několik základních pojmů a dále je zde zpracována problematika testování webových aplikací. Praktická část se zabývá návrhem a implementací webové aplikace pro simulaci činnosti deterministického konečného automatu. Dále jsou v práci popsány přístupy a nástroje pro testování, které byly v průběhu vývoje použity a také je zde zhodnocen jejich konkrétní přínos z hlediska kvality aplikace.

Cílem diplomové práce je vytvoření kvalitní webové aplikace pro simulaci deterministického konečného automatu s hlavním důrazem na animaci činnosti tohoto automatu. Druhotným cílem práce je vytvořit ucelený přehled dostupných přístupů a nástrojů pro testování webových aplikací a zároveň některé popsané nástroje při vývoji aplikace použít a poté zhodnotit jejich přínos na jednotlivá měřítka kvality vyvíjené aplikace.

2 TEORETICKÁ A REŠERŠNÍ ČÁST

2.1 VYMEZENÍ ZÁKLADNÍCH POJMŮ

Na úvod je vhodné vymezit několik základních pojmů, které se budou objevovat v dalších částech práce. Jejich správné definování, je nezbytným požadavkem pro další používání.

2.1.1 KONEČNÝ AUTOMAT

Konečný automat je definován jako uspořádaná pětice A = (Q, ∑, δ, q_0, F), kde

- Q je konečná neprázdná množina stavů
- ∑ je konečná neprázdná množina vstupních symbolů
- δ je přechodová funkce, δ: Q × ∑ → Q
- q_0 je počáteční stav, q_0 ∈ Q
- F je množina koncových stavů, F ⊆ Q

Slovo nad abecedou ∑ je libovolná konečná posloupnost prvků z ∑. Prázdné slovo, které značíme ε je takové slovo, které neobsahuje žádný vstupní symbol. Slovo je rozpoznáno konečným automatem, pokud po jeho přečtení skončí automat v některém z koncových stavů. Pokud se automat po přečtení slova nalézá ve stavu, který není koncový, jedná se o slovo nerozpoznatelné tímto automatem. Množina všech slov rozpoznatelných konečným automatem se nazývá jazyk rozpoznávaný konečným automatem.

Vzhledem k tomu, že definice přechodové funkce je poněkud méně srozumitelná pro konkrétní příklady, používají se jiné reprezentace, které jsou pro čtenáře mnohem srozumitelnější. Jsou to reprezentace pomocí grafu přechodů, tabulky přechodů a stavovým stromem. Více informací o konečných automatech je možné nalézt ve skriptech Michala Chytila [2].

2.1.2 WEBOVÁ APLIKACE

Webovou aplikací nazýváme aplikaci vykonávanou na webovém serveru, k níž je přistupováno přes počítačovou síť např. internet, nebo její uzavřenou modifikací

intranet. Uživatel přistupuje k aplikaci pomocí webového prohlížeče, který plní roli takzvaného tenkého klienta. To znamená, že prohlížeč slouží pouze pro zobrazení aplikace a pro předávání požadavků uživatele zpět serveru, přičemž vůbec nezná vnitřní logiku aplikace a server na něm není ani v nejmenším závislý. Součástí webové aplikace mohou být skripty psané v programovacím jazyce JavaScript, které jsou interpretovány přímo webovým prohlížečem a většinou slouží pro zajištění většího komfortu a interaktivity používané aplikace.

2.1.3 KVALITA APLIKACE

U aplikace jako celku lze sledovat mnoho znaků kvality. Můžeme je rozdělit na externí a interní znaky. Externí znaky vnímá uživatel aplikace a patří mezi ně například bezchybnost, použitelnost nebo efektivita. Naproti tomu interní znaky kvality uživatele v podstatě nezajímají vůbec, ovšem pro tvůrce aplikace mohou být velmi důležité. Patří mezi ně například udržovatelnost, flexibilita nebo znovupoužitelnost. Dále je potřeba popsat jednotlivé znaky konkrétněji. Následující část je převzata z knihy Dokonalý kód [3].

Externí znaky kvality

- Bezchybnost – Míra chyb ve specifikaci, návrhu a implementaci.
- Použitelnost – Lehkost použití a délka zaučovací doby nezbytné k běžnému užívání aplikace.
- Efektivita – Minimální spotřeba systémových prostředků, včetně paměti a doby zpracování.
- Spolehlivost – Schopnost aplikace vykonávat požadované funkce za určených podmínek, kdykoli je to zapotřebí s dlouhými přestávkami mezi poruchami.
- Integrita – Míra obrany aplikace proti neautorizovanému nebo nevhodnému přístupu k některým částem a datům.
- Robustnost – Pojem určující, jak dalece je aplikace schopna pokračovat v práci po zadání neplatného vstupu ve stresových podmínkách apod.

Interní znaky kvality

- Udržovatelnost – Určuje míru jednoduchosti, se kterou lze aplikaci upravit, rozšiřovat o nové funkce, zlepšovat výkon a opravovat chyby.
- Flexibilita – Určuje do jaké míry lze aplikaci modifikovat pro jiné způsoby využití, než pro které byla původně určena.
- Znovupoužitelnost – Míra, podle níž lze již hotové části aplikace použít v jiných aplikacích.
- Čitelnost – Míra jednoduchosti, s jakou můžeme kód číst a především mu dobře porozumět.
- Testovatelnost – Míra, do jaké můžeme aplikaci testovat na úrovni jednotek nebo na úrovni celé aplikace. Míra, do jaké můžeme ověřit, zda aplikace splňuje nastolené požadavky.
- Srozumitelnost – Míra jednoduchosti, s jakou jsme schopni systému porozumět nejen na organizační úrovni, ale také na úrovni jednotlivých příkazů zdrojového kódu.

Je zřejmé, že maximalizace některých znaků kvality, může vést ke konfliktům s pokusy o maximalizaci znaků jiných. Priorita znaků kvality by měla být stanovena již při specifikaci a následné analýze projektu, a tyto požadavky by měly být zohledněny při návrhu a implementaci aplikace.

2.1.4 TESTOVÁNÍ APLIKACE

Testováním aplikace nazýváme proces nebo sadu procesů, kterými aplikace opakovaně prochází za účelem ověření funkčnosti nebo hodnocení kvality. Testování by mělo být nedílnou součástí vlastního vývoje a dalších částí životního cyklu aplikace. Cílem testování je nalezení a oprava co největšího množství chyb, což vede ve výsledku ke kvalitnější aplikaci. Chybou nazýváme rozpor, mezi tím co aplikace dělat má a tím co ve skutečnosti dělá.

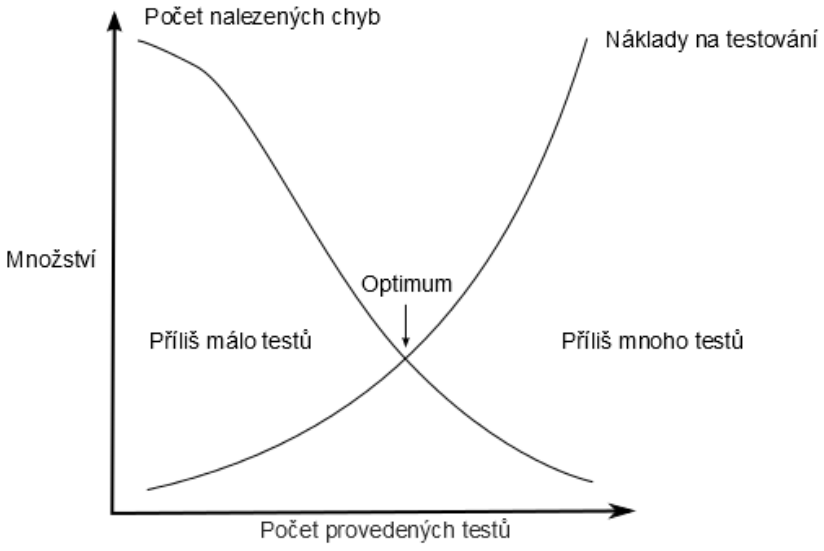
Mezi množstvím nalezených chyb a kvalitou aplikace platí nepřímá úměra, tedy čím je vyšší počet nalezených chyb, tím je nižší kvalita testované aplikace. Testováním nikdy nelze odhalit všechny chyby, neboť množina stavů, do kterých se aplikace může

dostat, je příliš veliká. Ale dá se předpokládat, že čím více chyb jsme objevili, tím víc neodhalených chyb v aplikaci ve skutečnosti je.

Po odhalení chyby je potřeba ohodnotit její závažnost, případně jestli se skutečně jedná o chybu. Toto hodnocení může mít mnoho různých hledisek. Výsledkem tohoto hodnocení může být i závěr, že chybu opravovat nebudeme. Například z důvodu přílišné nákladnosti opravy.

Pokud se rozhodneme chybu opravit, je potřeba po realizaci opravy aplikaci opět otestovat, abychom zjistili, zdali jsme chybu opravdu odstranili, případně, jestli jsme touto opravou nenarušili jinou část aplikace. Tímto iterativním postupem se snažíme dosáhnout co nejvyšší kvality aplikace. Je zřejmé, že pro různé znaky kvality je potřeba aplikaci testovat různými způsoby a v různých fázích vývoje aplikace. Ovšem zde platí jedno velmi důležité pravidlo a to, že všechny chyby se snažíme najít, a pokud možno opravit, co nejdříve je to možné, neboť pozdější objevení chyby může až několikrát prodražit její opravu. Příčina prodražení oprav chyb je způsobena tím, že na chybnou část projektu navazujeme další částí, které přizpůsobujeme právě oné chybné části. Proto je zřejmé, že čím dříve chybu objevíme, tím méně částí používá opravenou část projektu a tudíž náprava je mnohem jednodušší.

Ron Patton ve své knize o testování softwaru [4] uvádí, že každý softwarový projekt má určitou efektivní hladinu testování, od níž se již nevyplácí vytvářet další testy. Tato hranice je znázorněna na obrázku 2.1, který je převzat z této knihy.



Obrázek 2.1 – Efektivní hladina testování

2.2 VÝVOJOVÉ TESTOVÁNÍ

2.2.1 JEDNOTKOVÉ TESTOVÁNÍ

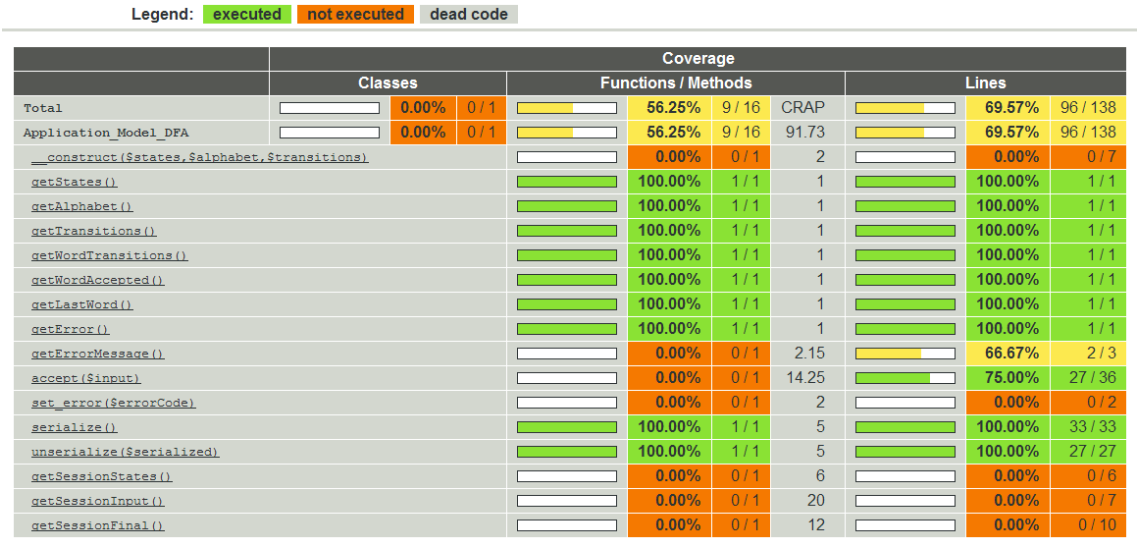
Při vývoji aplikací se někdy naráží na problém, kdy je vyvíjen nějaký souvislý celek z několika funkčních částí a otestování funkčnosti tohoto celku je možné až po vytvoření všech jednotlivých funkčních částí nebo po začlenění tohoto celku do aplikace. Typickou ukázkou tohoto problému jsou grafické aplikace. Funkčnost je ověřována až po vývoji uživatelského rozhraní, což je v rozporu s již zmíněným požadavkem nalezení chyby co nejdříve. Samozřejmě bychom mohli každou část aplikace otestovat vytvořením jakéhosi elementárního testu funkčnosti, ale tvořili bychom nadbytečný kód, který ve funkční aplikaci nebude mít žádný význam. Tento postup není zcela nejlepší a nejproduktivnější. Dalším problémem je, že některé stavy, jako například nefunkční připojení k databázi, se testují velmi špatně, i když bychom rádi věděli, jak na tento stav bude aplikace reagovat. Tyto problémy se snaží vyřešit metodika testování jednotek.

Metodika testování jednotek neboli unit testování se využívá přímo při vývoji aplikace. Cílem jednotkového testu je ověřit, zda komponenta plní funkci, kterou po ní požadujeme. Vývoj poté probíhá tímto způsobem. Nejprve implementujeme třídu, například kalkulačku a jednu její metodu, třeba sčítání. Dále vytvoříme test pro tuto metodu. To zjednodušeně uděláme tak, že metodě předáme určité vstupy a budeme očekávat určitý výstup. Když posléze spustíme testování, test zavolá testovanou metodu a vyhodnotí, jestli se její výstup shoduje s očekávaným výstupem. Jestliže se výstupy shodují, test je prohlášen za úspěšný. Pokud stejný není, je test označen jako neúspěšný a je třeba kód upravit, neboť neplní funkci, kterou bychom očekávali.

Testování jednotek tedy nepochybně ulehčuje vývoj tím, že k otestování komponent dochází prakticky okamžitě. Je ale možné namítnout, že zbytečně vytváříme neprodukční kód, jehož tvorba zvyšuje náklady. Ovšem tento kód nám zůstane a přináší nám mnohem víc pozitivních jevů. Testovací kód je například velmi dobrou dokumentací, neboť názorně specifikuje kontrakt metody. Další přínos je, že komponenty i celou aplikaci můžeme otestovat prakticky kdykoli, což může být vhodné, pokud provedeme v nějaké komponentě jistou změnu. Otestováním aplikace ihned zjistíme, zdali naše změna nemá nějaký nechtěný vedlejší efekt. Na druhou stranu

je nepochybné, že testování jednotek s sebou nese i jistá negativa. Zvyšuje se množství udržovaného kódu a testy také podléhají chybám. Vývojáři jsou nuceni přehodnotit svůj způsob práce a více přemýšlet. Někdy mohou testy vést k hlášení falešné chyby.

V neposlední řadě většina frameworků pro jednotkové testování podporuje analýzu pokrytí kódu. Řádek kódu, který je vykonán v rámci jednotkového testu nazveme pokrytý. Touto analýzou zjišťujeme, jak velkou část kódu jsme jednotkovými testy pokryli. Samozřejmě je žádoucí pokrýt co největší množství kódu, i když na druhou stranu pokrývat testy sto procent kódu může být často zbytečné. Analýza pokrytí kódu poskytuje detailní statistiky, jak jsou jednotlivé třídy a funkce pokryty. Příklad je možné vidět na obrázku 2.2.



Obrázek 2.2 – Statistika pokrytí kódu vygenerovaná frameworkem PHPUnit

Pokrytí kódu ovšem může dále posloužit i jako nástroj pro analýzu konkrétních řádků kódu. Představme si situaci, kdy vytvoříme jednotkové testy pro všechny možné situace, které pro použití metody připadají v úvahu. Pomocí funkce pokrytí kódu ovšem zjistíme, že několik řádků kódu zůstalo nepokrytých. Naskýtá se tedy otázka, zdali je nepokrytý kód zbytečný nebo jsme neotestovali všechny možné případy použití. Ukázka je uvedena na obrázku 2.3. Pomocí této analýzy můžeme zjistit, které konkrétní řádky kódu nebyly jednotkovými testy otestovány a proto je možné, že se v nich může nacházet chyba. Tato analýza může být velmi přínosná v případě, když zjistíme, že nějaká metoda v našem projektu nefunguje v reálném provozu tak jak má, přestože jsme

ji již pomocí jednotkového testu otestovali. Díky nástroji pokrytí kódu je možné nalézt nepokrytý kód, ve kterém bude pravděpodobně chyba, což může značně ušetřit čas. V této kapitole bylo čerpáno z knihy Professional PHP6 [5] a dokumentace frameworku PHPUnit [6].

```
60                :     public function accept($input) {
61                :         try{
62              1 :             $this->_error = -1;
63              1 :             $this->_wordTransitions = null;
64              1 :             $this->_lastWord = $input;
65              1 :             foreach ($this->_states as $state => $type) {
66              1 :                 if ($type=="start" || $type =="start-final") {
67              1 :                     $this->_currentState=$state;
68              1 :                     break;
69                :                 }
70              1 :             }
71              1 :             $end=strlen($input);
72              1 :             $this->_wordTransitions[0] = $this->_currentState;
73                :
74              1 :             for ($start=0;$start<$end;$start++) {
75              1 :                 $char=substr($input,$start,1);
76              1 :                 if ($this->_transitions[$this->_currentState]) {
77              1 :                     if ($this->_transitions[$this->_currentState][$char]) {
78              1 :                         $this->_currentState=$this->_transitions[$this->_currentState][$char];
79              1 :                         if (!$this->_transitions[$this->_currentState]) {
80              0 :                             $this->set_error(1);
81              0 :                             break;
82                :                         }
83              1 :                     }else{
84              0 :                         $this->set_error(0);
85              0 :                         break;
86                :                     }
87              1 :                 }else{
88              0 :                     $this->set_error(1);
89              0 :                     break;
90                :                 }
91              1 :                 $this->_wordTransitions[$start+1] = $this->_currentState;
92              1 :             }
```

Obrázek 2.3 – Pokrytí jednotlivých řádků kódu

2.2.2 JEDNOTKOVÉ TESTOVÁNÍ PRO PHP

Enhance PHP

Enhance PHP je jednoduchý framework pro jednotkové testování. Cílem tohoto projektu je maximální jednoduchost – celý framework je obsažen v jednom souboru o velikosti 5 kB. Framework je využitelný pro velmi malé projekty nebo pro vývojáře, kteří nemají s jednotkovým testováním zkušenosti. Informace o tomto frameworku byly čerpány z jeho domovské stránky [7].

SimpleTest

SimpleTest je testovací framework, který je podporován vývojovým prostředím Eclipse. V současné verzi 1.1.0. je použitelný pro základní jednotkové testování. V budoucnu bude zajímavé tento projekt sledovat, neboť autoři slibují implementaci dalších zajímavých funkcí, jako například podporu Selenia nebo znázornění pokrytí

kódu. Informace o tomto frameworku byly čerpány z domovské stránky tohoto projektu [8].

PHPUnit

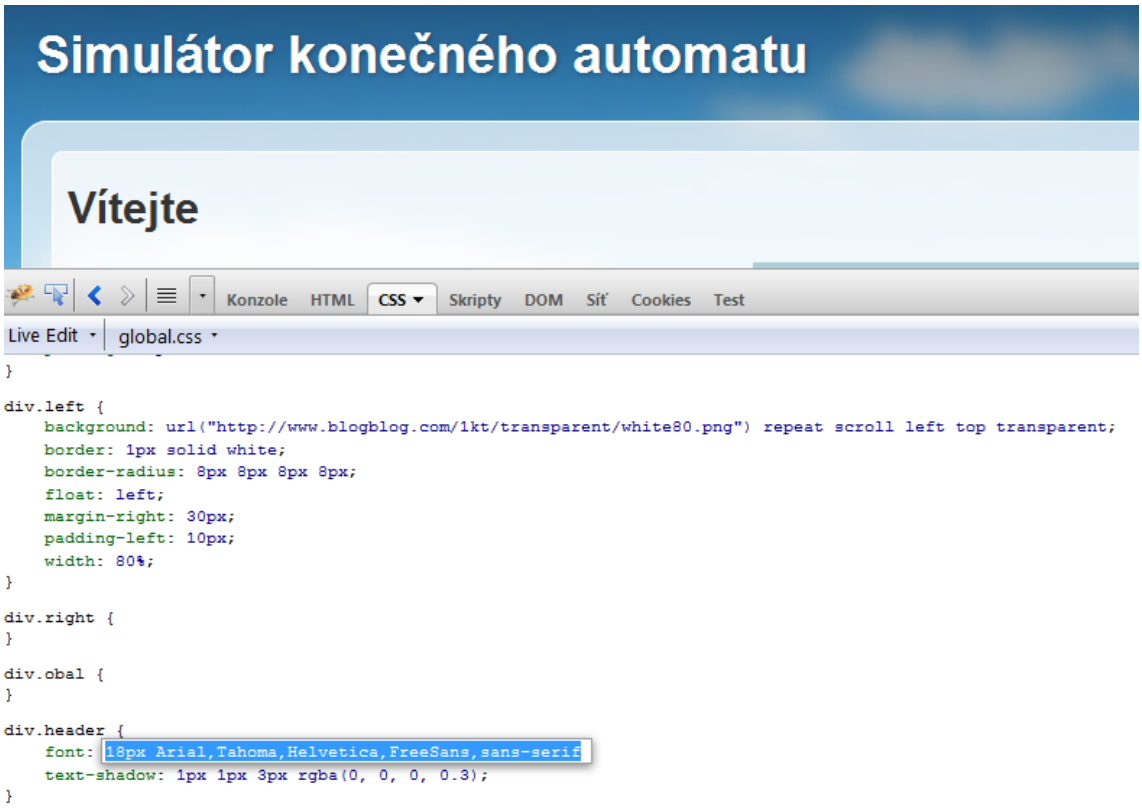
PHPUnit je robustní framework pro jednotkové testovaní a dá se říci, že v současné době se jedná o nejpoužívanější a nejkvalitnější nástroj. Poskytuje velké množství možností způsobů testování a zároveň umožňuje integraci s jinými používanými programy pro testování. Obsahuje také velmi kvalitní a rozsáhlou dokumentaci. Framework je podporován vývojovým prostředím NetBeans a je také velmi dobře dokumentovaný [6].

2.2.3 TESTOVÁNÍ VZHLEDU

Při vývoji webové aplikace hraje také velmi výraznou roli v časové náročnosti tvorba a úprava vzhledu. Vzhledem k používaným technologiím HTML a CSS vývoj vzhledu často probíhá metodou pokus-omyl. Vývojář upraví zdrojové kódy aplikace a poté musí stránku ve webovém prohlížeči znovu načíst a zkontrolovat, jak se změny projevily. Tímto postupem se dříve či později dospěje k požadovanému výsledku. Ovšem pokud má vývojář k dispozici nástroj, který vzhled stránky upraví automaticky ihned po jakékoliv úpravě kódu, vývoj vzhledu se několikanásobně urychlí.

Firebug

Firebug je plugin pro internetový prohlížeč Mozilla Firefox, který výrazným způsobem usnadňuje vývoj a kontrolu vzhledu. Umožňuje editovat zdrojový kód stránky, přičemž prohlížeč změny okamžitě realizuje. Plugin také obsahuje takzvaný inspekční mód. Ten funguje tak, že kurzorem najedeme na jakýkoliv objekt webové stránky a plugin okamžitě na webové stránce vyznačí, který objekt jsme vybrali, a ve zdrojových kódech stránky zvýrazní řádky, které se vztahují k vybranému objektu. Tímto způsobem je zajištěna velmi jednoduchá a rychlá kontrola vzhledu za účelem hledání chyb. Ukázku tohoto nástroje je možné vidět na obrázku 2.4. Více informací je možné nalézt na webových stránkách projektu [9].



Obrázek 2.4 – Ukázka nástroje FireBug

2.3 FUNKČNÍ TESTOVÁNÍ

Funkčním testováním se snažíme ověřit správnost funkčnosti jednotlivých prvků webové aplikace. S ohledem na testované prvky můžeme funkční testování rozdělit do několika podskupin. V této kapitole bylo čerpáno ze článku o funkčním testování webových aplikací [10].

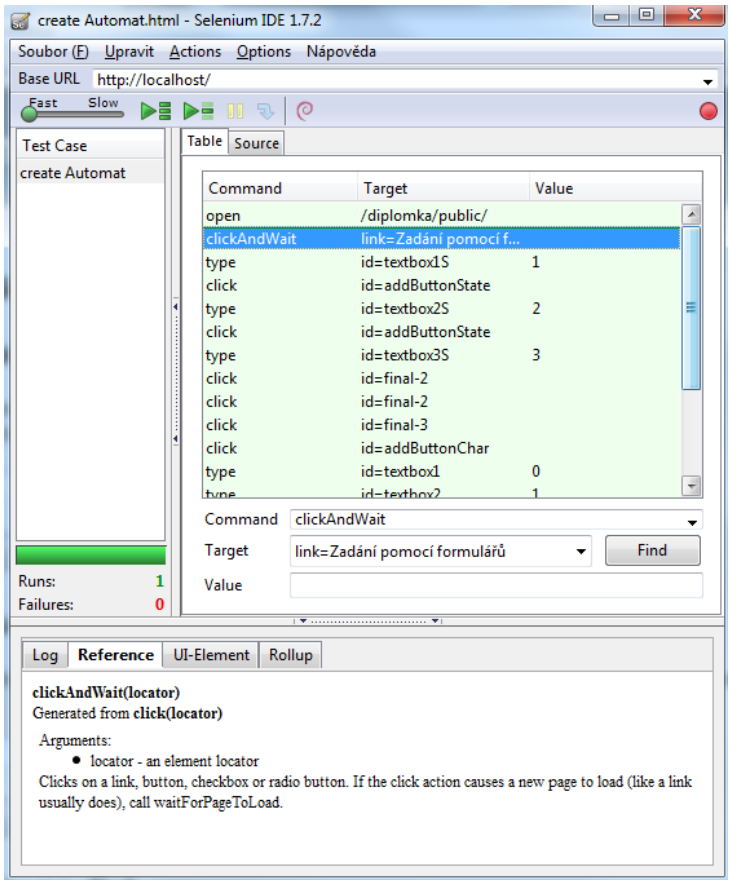
2.3.1 KONTROLA KONKRÉTNÍCH PRVKŮ

V této podskupině funkčního testování je cílem otestovat výskyt a obsah nějakého konkrétního prvku webové stránky. Tento způsob testování se využívá v situaci, kdy je potřeba v aplikaci provést několik uživatelských kroků a poté zkontrolovat, jaký je výsledek těchto kroků. Nástroje pro testování konkrétních prvků umožňují tuto posloupnost uživatelských kroků zaznamenat a test je poté možné automatizovat, proto se někdy tomuto způsobu testování říká automatické.

Selenium

Selenium je rodina testovacích nástrojů pro automatizované testování webových aplikací. Základním programem je Selenium IDE, který slouží pro zaznamenávání a automatizaci testů na lokálním počítači. Dále jsou k dispozici programy, které umožňují testy vytvářet programově v různých programovacích jazycích a také programy, které umožňují tyto testy spouštět na vzdálených počítačích.

Selenium IDE je plugin pro internetový prohlížeč Mozilla Firefox. V tomto prostředí je možné vytvořit jakýkoliv testovací scénář založený na použití webového prohlížeče. To v podstatě znamená, že můžeme zaznamenat jakoukoliv činnost uživatele, kterou poté můžeme automaticky a zrychleně provádět znovu. Obrovskou výhodou použití tohoto produktu je, že jednotlivé testy je možné uložit a v případě potřeby je opět znovu použít, přičemž je možné nastavit rychlost vykonání jednotlivých kroků testu. Použitím tohoto produktu při vývojářském testování je možné značným způsobem urychlit vývoj aplikace.



Obrázek 2.5 – Ukázka pluginu Selenium IDE

Automatizované testy lze také vytvářet programově. Podporovány jsou programovací jazyky Java, C#, Python, Ruby, Perl a PHP. Programové testy jsou poté vykonány nástrojem Selenium WebDriver. Tímto způsobem je možné provádět testy na velmi detailní úrovni, ovšem je potřeba si uvědomit, že daní za tento přínos jsou relativně vysoké náklady na vytvoření těchto testů. Ukázka programově vytvořeného testu v programovacím jazyce Java je uvedena na obrázku 2.6. V této kapitole bylo čerpáno z domovské stránky projektu [11], zde je také možné najít další informace a dokumentaci.

```
package org.openqa.selenium.example;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.ExpectedCondition;
import org.openqa.selenium.support.ui.WebDriverWait;

public class Selenium2Priklad  {
    public static void main(String[] args) {
        // Vytvoreni instance prohlizece Firefox
        WebDriver driver = new FirefoxDriver();
        // Prechod na stranku www.google.com
        driver.get("http://www.google.com");
        // Nalezneme textovy input podle jmena
        WebElement element = driver.findElement(By.name("q"));
        // Vlozime dotaz pro vyhledani
        element.sendKeys("Selenium");
        // Odeslani formulare
        element.submit();
        // Overeni nazvu stranky
        System.out.println("Nazev stranky je: " + driver.getTitle());
        // Zpozdeni 10 sekund
        (new WebDriverWait(driver, 10)).until(new ExpectedCondition<Boolean>() {
            public Boolean apply(WebDriver d) {
                return d.getTitle().toLowerCase().startsWith("Selenium");
            }
        });
        // Meli bychom videt: "Selenium - Google Search"
        System.out.println("Nazev stranky je: " + driver.getTitle());
        // Zavreni prohlizece
        driver.quit();
    }
```

Obrázek 2.6 – Ukázka programově vytvořeného testu v jazyce Java

Windmill

Windmill je aplikace vytvořená v programovacích jazycích Python a JavaScript. Díky tomu je funkční ve většině používaných prohlížečů. Stejně jako Selenium IDE umožňuje automatizovat a ukládat testy. Navíc díky podpoře více prohlížečů je možné jeden stejný test provádět různými prohlížeči. Daní za tuto možnost je relativně složitá instalace a nepříliš komfortní ovládání pomocí konzole. Umožňuje také testy vytvářet pomocí programovacího jazyka JavaScript. Informace o této aplikaci byly čerpány z domovské stránky projektu [12].

2.3.2 KONTROLA ODKAZŮ

Hovoříme-li o kontrole odkazů, je naším cílem otestovat, jestli všechny odkazy dané webové aplikace odkazují na správný cíl, případně jestli je jejich cíl dostupný. Tento typ testování je vhodné provádět průběžně, ať už s ohledem na externí odkazy nebo na dynamicky přidávané soubory. Pokud jsou všechny odkazy v naší aplikaci funkční, aplikace se uživatelům jeví jako udržovaná a důvěryhodná.

Xenu

Xenu je grafická aplikace pro kontrolu funkčnosti odkazů na webové stránce. Aplikace má relativně velké množství nastavení. Umožňuje testovat několik stránek najednou, určit zda chceme kontrolovat i externí odkazy a zároveň do jaké hloubky chceme odkazy testovat. Zajímavou možností je také nastavení počtu paralelních procesů neboli vláken, která mohou probíhat zároveň. Aplikace je bohužel implementována pro operační systémy Windows 95 a novější, což snižuje potenciál použitelnosti na jiných platformách.

W3C Link Checker

W3C Link Checker je webová aplikace vytvořená v programovacím jazyce Perl s veřejně přístupným zdrojovým kódem. Aplikaci je také možné nainstalovat na lokálním počítači. Aplikace kromě funkčnosti odkazů také kontroluje, zdali nedochází u odkazů k přesměrování. Také je zde možné nastavit hloubku testovaných odkazů. Nevýhodou této aplikace je, že testuje pouze odkazy na HTML dokumenty.

LinkChecker

LinkChecker je multiplatformní aplikace vytvořená v programovacím jazyce Python. Je dostupná v konzolové i grafické verzi. Stejně jako předchozí aplikace umožňuje zvolit hloubku zanoření při kontrole odkazů. Velkou výhodou této aplikace je, že u všech upozornění a chyb srozumitelně interpretuje, odkud se na tyto nefunkční odkazy přistupovalo, a k číselnému kódu chyb přidává slovní popis chyb a doporučený způsob nápravy, což velmi usnadňuje orientaci ve výsledcích a zároveň zjednodušuje opravu.

2.3.3 HTML A CSS VALIDACE

Validátory slouží k ověření, zdali je kód webové stránky vytvořen podle doporučené specifikace konsorcia W3C. Pokud tak stránka vytvořena není, nemusí to znamenat vážnou chybu, ale z odborného hlediska na ni mohou ostatní vývojáři pohlížet jako nekvalitní. Validace se provádí hlavně kvůli možnosti, že různé prohlížeče budou chybný kód interpretovat různě, čemuž se snažíme zamezit. Bohužel v současnosti je situace taková, že i validní stránky mohou a často i bývají interpretovány v některých prohlížečích jinak, než by být interpretovány měly. Proto pouhá validace jednotný vzhled v prohlížečích nezaručí.

Na webové stránce konsorcia W3C – www.w3.org jsou k dispozici webové aplikace pro HTML i CSS validaci. Nevýhoda těchto aplikací spočívá v tom, že je vždy zkontrolována pouze jedna stránka testované webové aplikace. Abychom mohli prohlásit, že je celá aplikace validní, je potřeba tuto validaci provést nad všemi stránkami, které naše aplikace obsahuje. Tento problém se týká především HTML validace, neboť dnešním standardem bývá všechny CSS styly sdružit do jednoho souboru, který je pro každou stránku aplikace stejný.

WDG HTML Validator

Tato webová aplikace se snaží odstranit nedostatek validátoru W3C, neboť nabízí možnost validovat celý web, ovšem s omezením počtu validovaných stránek hodnotou 100. Což opět u velkého webu může být velmi omezující. Řešením by mohlo být rozdělení validovaných částí na podskupiny a ty validovat zvlášť.

2.3.4 KONTROLA FORMULÁŘŮ

Hovoříme-li o kontrole formuláře, nabízí se dva pohledy. První pohled se týká ověření funkčnosti formuláře. Naším cílem je prakticky otestovat, jestli formulář pracuje tak, jak by měl podle specifikace. Tedy validní data přijme a nevalidní data odmítne. Tento test lze provést funkčním testováním, které je již popsáno výše.

Druhý pohled se týká testování situací, které by teoreticky neměly nastat, ale nelze je vyloučit. Například použijeme-li ve formuláři objekt typu radio a nějakou hodnotu z něj vybereme jako standardní. Co když se uživateli nějakým způsobem povede formulář odeslat tak, že v objektu radio nebude vybrán žádný prvek? Jistě by nebylo žádoucí, aby takováto triviální chyba vedla k pádu aplikace, proto je vhodné otestovat formulář i pro takovýto scénář.

Web Develeper tool

Web Developer Tool je rozšíření internetového prohlížeče Mozilla Firefox. Umožňuje přehledně zobrazit veškeré informace o formuláři na aktuální webové stránce. Dále umožňuje změnit odesílací metodu formuláře, povolit zakázané prvky formuláře, odstranit zaškrtnuté přepínače, nebo odstranit omezení délky v objektech.

2.3.5 KONTROLA COOKIES

Cookie je malé množství dat uchované v textovém souboru uložené na uživatelském počítači webovým serverem. Zpravidla se užívají pro ukládání unikátních dat pro jednotlivé uživatele. Pokud aplikace cookies používá, je dobré mít nějaký nástroj, kterým bychom mohli kontrolovat jejich zápis a stav. Případně otestovat, jak bude aplikace fungovat, pokud bude mít klient použití cookies zakázáno.

Web Developer tool

Toto již zmíněné rozšíření umožňuje jednoduchým způsobem spravovat cookies – zobrazení, přidání, úpravy a smazání. Umožňuje také cookies explicitně zakázat.

Firecookie

Firecookie je rozšíření doplňku Firebug pro internetový prohlížeč Mozilla Firefox. Umožňuje spravovat cookies a také je explicitně zakázat. Ve srovnání

s přechozím nástrojem je použití Firecookie pohodlnější a jeho uživatelské rozhraní je přehlednější.

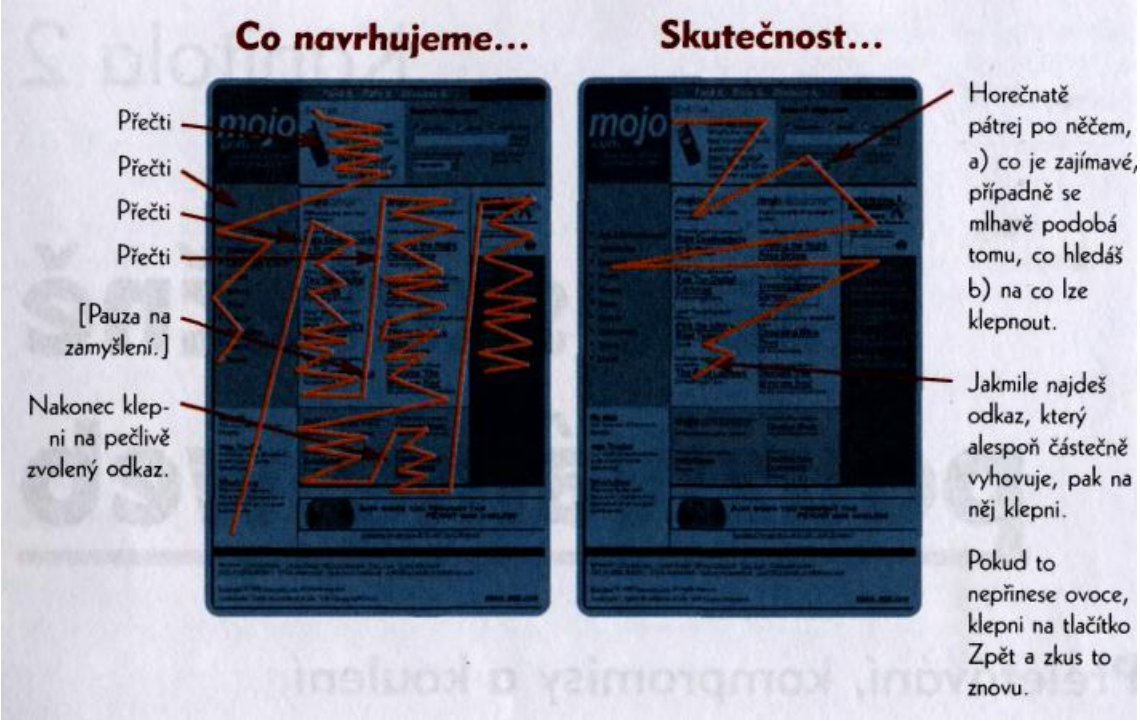
2.4 TESTOVÁNÍ POUŽITELNOSTI

Testování použitelnosti je oblast zabývající se rozhraním webové aplikace ve smyslu přehlednosti a použitelnosti. Požadavek na maximální přehlednost a použitelnost vznikl především u komerčních aplikací, jejichž cílem je, aby stránku navštívilo a použilo co největší množství uživatelů, díky kterým majitelé získají nějaký profit.

Testování použitelnosti analyzuje chování uživatelů při používání aplikací, s cílem najít a vymezit základní pravidla a chyby při tvorbě rozhraní webových aplikací. Bohužel i při dodržení základních pravidel pro tvorbu uživatelského rozhraní se může stát, že aplikace bude pro uživatele nevyhovující. Je proto vhodné aplikaci otestovat na určitém vzorku uživatelů.

Test použitelnosti aplikace se skládá ze tří fází. V první části je potřeba specifikovat základní úkoly, které bude typický uživatel v aplikaci vykonávat a na základě této specifikace vytvořit testovací scénář. Tedy posloupnost úkolů, které bude mít uživatel za úkol vykonat. Další částí testu použitelnosti je samotné testování. Mělo by probíhat s uživatelem, který aplikaci nezná a podobné testování dosud neabsolvoval. Pozorovatel uživateli postupně dává úkoly a sleduje, jak si uživatel poradí s jejich plněním. Pro pozorovatele jsou nejzajímavější místa testu ta, kde má uživatel se splněním úkolu problém nebo váhá, jestli je jeho postup správný. Celý tento test použitelnosti je možné zaznamenat nějakým programem na zaznamenávání obrazovky a mikrofonu, ovšem neměl by to být hlavní zdroj informací. Celý test by měl trvat maximálně jednu hodinu. Po jeho skončení by měl pozorovatel nebo pozorovatelé vyhodnotit, kde měl uživatel největší problémy, neboť fakt, že uživatel měl s úkolem na některém místě aplikace problém, signalizuje, že uživatelské rozhraní aplikace není navržené správně. Stejný test použitelnosti je vhodné vykonat asi se třemi až pěti uživateli. Po dokončení těchto testů je zde poslední fáze – fáze vyhodnocení. Zde je potřeba shrnout veškeré informace získané od uživatelů. Z těchto chyb jsou vybrány ty, které se opakují nejčastěji. Ty jsou poté přesně zdokumentovány a předány vývojářům,

jejichž úkolem je tyto chyby opravit. Po opravě je vhodné podobný test opakovat s novým vzorkem uživatelů. Pokud se dříve objevené chyby již neobjeví, je to signál, že jejich oprava proběhla správně. Na druhou stranu mohou při tomto dalším testování vyplout na povrch další chyby, které se dříve neobjevily, a je na zvážení pozorovatelů, zdali se jedná opravdu o závažné chyby nebo jen o dílčí chyby, jejichž oprava není nutná. V této kapitole bylo čerpáno z přednášky Testování použitelnosti [13] a knihy Nenuťte uživatele přemýšlet [14], ze které je také převzat obrázek 2.7, na kterém je možné vidět rozdíl mezi chováním uživatele očekávaným vývojáři a chováním reálného uživatele.



Obrázek 2.7 – Rozdíl mezi očekávaným a reálným chováním uživatele

2.5 TESTOVÁNÍ KOMPATIBILITY

Vzhledem k množství zařízení a prohlížečů, ze kterých mohou uživatelé na webovou aplikaci přistupovat, je přirozené otestovat přístupnost aplikace i z těchto hledisek. Základním testováním kompatibility je testování v různých webových prohlížečích. Cílem je ověřit, zde je vzhled ve všech aplikacích stejný resp. velmi podobný.

2.5.1 TESTOVÁNÍ V PROHLÍŽEČÍCH

Browsershots

Browsershots je webová aplikace, která umožňuje kontrolu zobrazení webové aplikace v různých prohlížečích a na různých platformách. Vstupem je odkaz na webovou stránku, kterou chceme testovat, a tedy námi testovaná aplikace musí být online na internetu. Pro testování aplikace na lokálním serveru je aplikace browsershots nepoužitelná. Aplikace jako výsledek vytváří obrázky ve formátu png, které obsahují vzhled zadané aplikace v konkrétním prohlížeči. Obrovskou výhodou aplikace je velké množství prohlížečů, jejich verzí a dále různé verze operačních systémů.

Netrenderer

Netrenderer je webová aplikace pro kontrolu vzhledu webové aplikace v prohlížečích Internet Explorer. Funguje na podobném principu jako browsershots. V současné době je možné ji využít pro otestování aplikace v prohlížečích Internet Explorer ve verzích 5.5 až 9.

Spoon.net

Spoon.net umožňuje spouštět aplikace z internetu bez instalace na lokálním počítači, přičemž nastavení se ukládá do cloudu. Tato aplikace mimo jiné nabízí řešení pro ladění aplikace v jednotlivých prohlížečích, neboť umožňuje spustit většinu v současnosti používaných webových prohlížečů v různých verzích. V případě, že by vývojář chtěl testovat svou webovou aplikaci ve všech těchto prohlížečích, znamenalo by to, že by je musel na svůj lokální počítač nainstalovat. To by bylo ovšem zbytečně časově náročné a na použití velmi nepraktické. Naproti tomu při použití aplikace Spoon.net si pouze vybere prohlížeč, který chce používat a ten do několika desítek sekund nastartuje. Nevýhodou tohoto řešení je, že se jedná o komerční službu, která ve verzi zdarma umožňuje využívat spuštěný program pouze po dobu dvou minut denně. Další použití je zpoplatněno.

2.5.2 TESTOVÁNÍ NA MOBILNÍCH PLATFORMÁCH

MobiReady

MobiReady je webová aplikace sloužící k otestování webové aplikace určené pro mobilní zařízení. U aplikace je testováno, zdali dodržuje standardy a také zdali dodržuje zásady doporučené pro mobilní aplikace, například požadavek na malou velikost stránky. Výsledkem testování je jakési skóre, které určuje použitelnost aplikace na mobilních zařízeních. Pokud testovací aplikace nalezne v testované aplikaci nějaké chyby, ve výsledcích jsou konkrétně popsány i s doporučeným řešením jejich opravy. Je zde k dispozici také softwarový emulátor, který obsahuje několik základních typů mobilních telefonů. V tomto emulátoru je možné testovanou aplikaci prohlížet. Aplikace také vytváří graf, který prezentuje, jak dlouho bude trvat načtení stránky na různých připojeních, což je také velmi přínosná informace.

W3C mobileOK Checker

W3C mobileOK Checker je webová aplikace vytvořená konsorciem W3C sloužící pro otestování webové aplikace, zdali je vhodná pro prohlížení mobilním zařízením. Aplikace poskytuje podobnou analýzu týkající se zdrojového kódu jako předchozí aplikace, ovšem není tolik přehledná.

Opera Mini Simulator

Opera Mini Simulator je webová aplikace, která simuluje funkci mobilního prohlížeče Opera na virtuálním mobilním telefonu. Je tedy možné funkčnost aplikace otestovat přímo v tomto emulátoru. Po srovnání se skutečnou aplikací na reálném mobilním telefonu je možné konstatovat, že simulátor emuluje reálnou aplikaci velmi věrně. Jako nedostatek se může jevit, že není možné měnit rozlišení displeje virtuálního telefonu, neboť by bylo vhodné zobrazení aplikace otestovat na nejčastěji používaných rozlišeních displejů.

2.6 VÝKONNOSTNÍ TESTOVÁNÍ

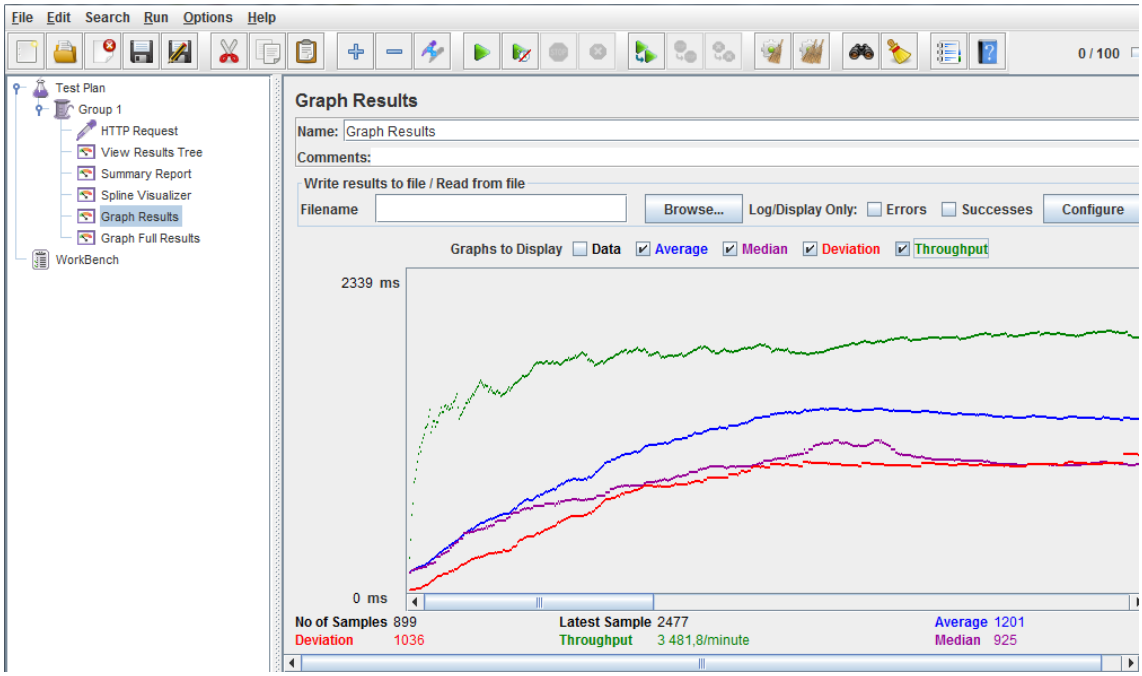
Výkonnostní testování se zabývá ověřováním, zdali aplikace splňuje specifikované výkonnostní parametry, jako například dobu odezvy nebo dostupnost. Obvykle se provádí simulováním simultánně přistupujících stovek, někdy i tisíců

uživatelů během určitého časového intervalu. Informace o přístupech jsou zaznamenány a poté je analyzováno, v jaké míře jsou čerpány systémové prostředky.

Vzhledem k faktu, že není možné dopředu odhadnout, jaké množství uživatelů bude aplikaci současně využívat v reálném provozu, je potřeba klást na výkonnostní testování patřičný důraz. Je také vhodné výkon a zatížení aplikace sledovat průběžně a případně aplikaci či hardware předimenzovat dle aktuálních požadavků.

Apache JMeter

Apache JMeter je desktopová aplikace vytvořená v programovacím jazyce Java, sloužící k výkonnostnímu testování. Umožňuje testovat různé služby např. HTTP, HTTPS, SOAP, FTP, POP3, IMAP a také databázové příkazy. Dále umožňuje detailně specifikovat další parametry testů např. počet aktivních uživatelů, rychlost nárůstu dalších uživatelů a počet požadavků, které uživatelé vytvoří. Tímto způsobem je možné vytvořit libovolné množství testů, které lze poté spouštět zvlášť nebo současně. Velkým kladem aplikace je velké množství možností interpretace výsledků, které je možné znázornit například pomocí tabulek, stromů, grafů či jiných komponent aplikace. Na obrázku 2.8 je znázorněn výsledek testu HTTP požadavku pomocí grafu, kde postupně přibývali aktivní uživatelé. Informace o aplikaci byly čerpány z domovské stránky tohoto projektu [15].



Obrázek 2.8 – Ukázka programu Apache JMeter

2.6.1 LOAD TESTING

Load testing bychom mohli přeložit jako zatěžovací testování, proto je mnohdy považováno za ekvivalent výkonnostního testování, od kterého se ovšem liší požadavkem, aby se výkon aplikace hodnotil na základě předem definované úrovně zatížení. Má za cíl měřit dobu potřebnou k vykonání některých úloh a funkcí za předem daných podmínek, které by měly obsahovat minimální konfiguraci a maximální zatížení běžící aplikace. Stejně jako v předchozím případě je prováděna simulace současně přistupujících uživatelů a pokud jsou překročeny předdefinované časové limity, je vygenerováno chybové hlášení.

Tímto testováním se ověřuje výkon aplikace v očekávaném provozu. Pokud bychom například testovali internetový obchod, je zřejmé, že většina uživatelů si bude produkty pouze prohlížet a oproti nim bude mnohem menší množství uživatelů opravdu nakupovat. Proto je třeba na základě těchto faktů při simulování zátěže generovat uživatele v poměru co nejbližším očekávané realitě.

2.6.2 STRESS TESTING

Stress testing se provádí tak, že je aplikace nebo komponenta vystavena zátěži, která se blíží nebo přesahuje její limity. Díky tomu je možné zjistit, jak bude aplikace reagovat při překročení únosné zátěže. Například jestli dojde k selhání nebo aplikace bude schopna se zotavit. Tento způsob testování se liší od předchozích dvou tím, že se testuje za únosnou hranicí požadavků, zatímco u předchozích způsobů se testuje normální, lepé řečeno rozumná uživatelská aktivita.

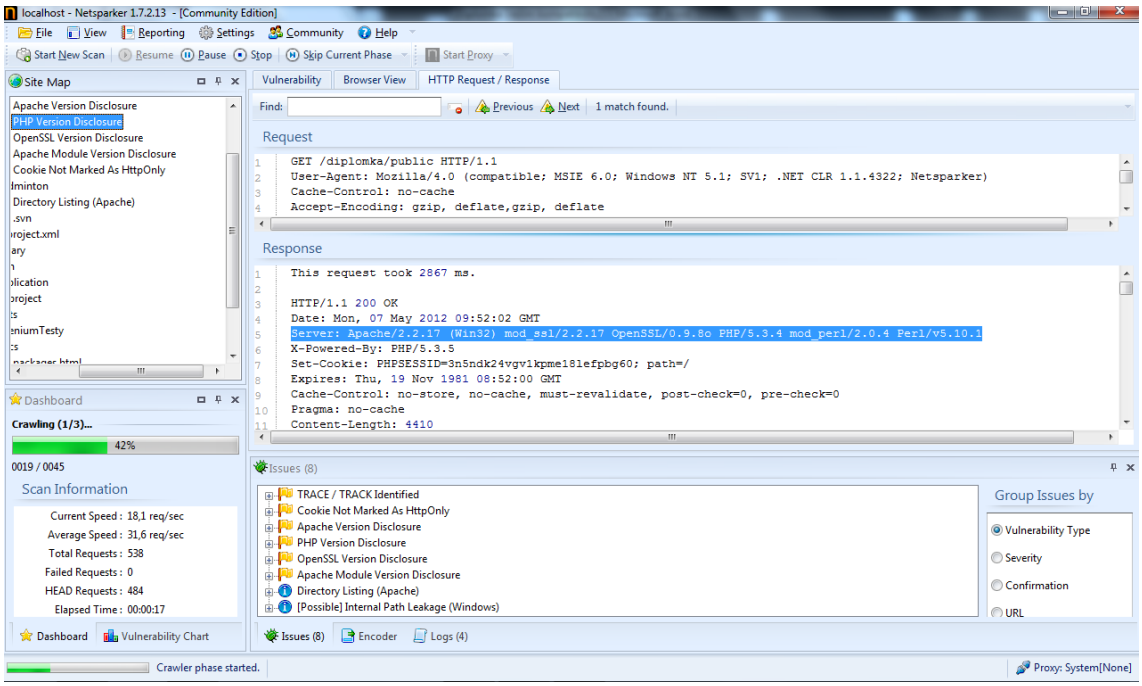
2.7 BEZPEČNOSTNÍ TESTOVÁNÍ

Bezpečnostní testování má za úkol ověřit celkovou schopnost ochrany webové aplikace proti nepovolenému přístupu neautorizovaných uživatelů, systémových prostředků proti nepovolenému použití a zároveň umožnit autorizovaným uživatelům používat některé prostředky a služby. Ochrana aplikace musí poskytnout ochranné mechanismy, schopné zabránit nebo omezit škody způsobené neoprávněnými průniky, přičemž náklady na tuto ochranu by měly být výrazně nižší než náklady na opravu škod, které mohou být způsobeny neoprávněným průnikem.

Chyby ovlivňující bezpečnost mohou být obsaženy v kódu aplikace nebo i v jiných hardwarových či softwarových komponentách, což činí bezpečnost jedním z nejpalčivějších problémů webových aplikací. Ideální je vývoj konzultovat s bezpečnostním expertem. Bezpečnostní chyby se nejčastěji vyhledávají pomocí bezpečnostních skenerů, které na aplikaci testují nejběžnější způsoby útoků.

Netsparker

Netsparket je takzvaný bezpečnostní skener pro webové aplikace. Jeho cílem je na testované aplikaci nalézt bezpečnostní chyby s hlavním důrazem na SQL injection a XSS útok. Testování je prováděno na každé jednotlivé stránce aplikace, aby bylo zajištěno komplexní otestování aplikace. Použití aplikace je relativně jednoduché. Stačí zadat URL adresu testované aplikace a spustit test. V nastavení aplikace je možné upravit některé parametry testování např. hlavičku používaného agenta. Vzhledem k rozsáhlosti testování, závislosti na výkonu serveru a také propustnosti sítě je testování touto aplikací relativně časově náročné. Testování jednoduché aplikace na lokálním serveru trvalo hodinu a půl. Pokud bychom testovali aplikaci na vzdáleném serveru s nižším výkonem a menší propustností sítě, může testování trvat i několik desítek hodin. Více informací o této aplikaci je možné nalézt na domovské stránce tohoto projektu [16].



Obrázek 2.9 – Ukázka programu Netsparker

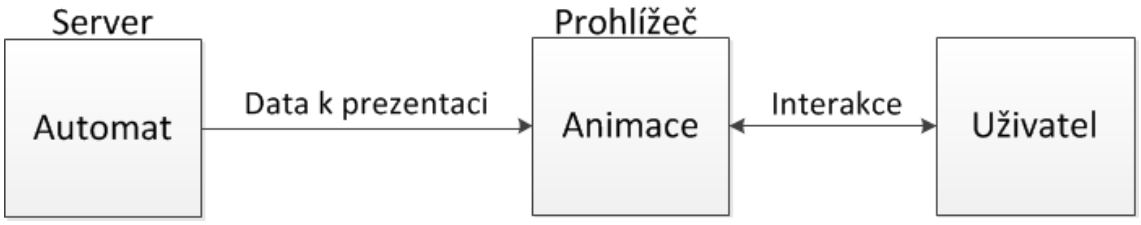
3 ANALÝZA A NÁVRH APLIKACE

V této kapitole je popsán obecný návrh aplikace, výběr technologií a výčet navrhovaných vylepšení aplikace, oproti původnímu stavu.

3.1 ANALÝZA PROJEKTU

V rámci zadání magisterského projektu byl kladen hlavní důraz na simulaci činnosti automatu a bylo tedy vhodné vytvořit interaktivní animaci činnosti konečného automatu. Zejména s ohledem na zátěž webového serveru a rychlost odezvy se jako nejvhodnější způsob zajištění této interaktivní animace jevila varianta, kde by animace a interakce s uživatelem byla zajištěna na straně klienta, ideálně bez nutnosti komunikace s webovým serverem.

Fyzické vyhodnocení průchodu zadaného slova automatem by bylo možné také realizovat pouze na straně klienta, ovšem z hlediska možného použití modelu konečného automatu se tato varianta jako nejvhodnější nejeví. Mnohem přijatelnější se zdá varianta, kdy vyhodnocení slova, jakožto aplikační část, bude probíhat na straně serveru a samotná animace, jakožto prezentační část, bude zajištěna na straně klienta. Tímto rozdělením na aplikační a prezentační část získá aplikace na flexibilitě. Například kdyby bylo potřeba vytvořit další způsob animace, třeba pomocí jiných technologií, zůstala by aplikační část stejná. Pouze by se přidala další prezentační část a uživatel by mezi těmito prezentačními způsoby mohl přepínat. Tyto prezentační části bychom také mohli měnit nebo nastavovat programově například na základě platforem klientů, nebo jejich rozlišení.



Obrázek 3.1 – Základní schéma návrhu aplikace

Aplikace by tedy měla fungovat tak, že uživatel nějakým způsobem zadá konkrétní automat a slovo, které chce automatem projít. Vytvoření automatu a vyhodnocení slova proběhne na serveru. Poté dojde k odeslání všech potřebných informací ke klientovi, kde by mělo dojít k jejich správné prezentaci.

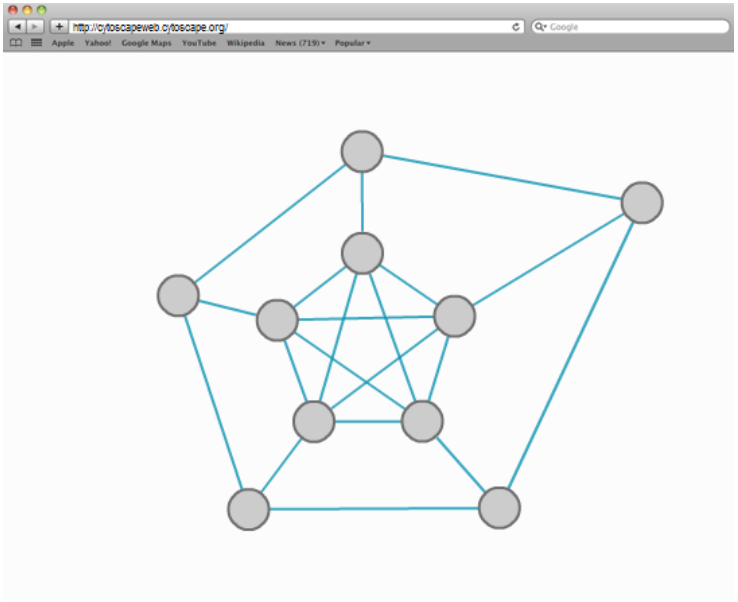
3.2 VÝBĚR TECHNOLOGIÍ

Jako skriptovací jazyk pro serverovou část byl zvolen jazyk PHP[1], zejména kvůli svému masivnímu rozšíření, dostupnosti a velkému množství bezplatných hostingů, které jej podporují. Dále byl pro vývoj serverové části zvolen Zend framework [17], což je objektově orientovaný aplikační framework implementovaný v PHP. Používá architekturu MVC, čímž vede programátora k tomu, aby aplikaci rozumně strukturoval.

Pro skriptování ve webovém prohlížeči byl zvolen programovací jazyk JavaScript a spolu s ním bude využita knihovna jQuery [18], která zjednodušuje použití tohoto jazyka.

Nejdůležitější částí aplikace by měla být animace, a proto byla výběru technologie věnována patřičná pozornost. Bylo vyzkoušeno několik knihoven pro grafickou reprezentaci dat. Jako jedna z nejzajímavějších se ukázala knihovna Protovis[2], která obsahuje mnoho zajímavých možností, jak data graficky reprezentovat, ovšem pro použití v této práci se ukázala jako nepříliš vhodná. Pro animační část aplikace byla nakonec zvolena knihovna Cytoscape Web, která vykresluje grafiku prostřednictvím programu Flash od firmy Adobe. Jedná se tedy o vektorovou grafiku. Autoři uvádějí [19], že je knihovna vhodná pro tvorbu interaktivních síťových grafů na webu. Knihovna je ovládána pomocí programovacího jazyka JavaScript.

[1] Hypertext Preprocessor – více informací je možné nalézt na http://www.php.net/
[2] Více informací o knihovně Protovis je možné nalézt na http://mbostock.github.com/protovis/



Obrázek 3.2 – Ukázka grafické knihovny Cytoscape Web

3.3 NÁVRH APLIKACE

Aplikace by měla umožnit uživateli vytvořit jakýkoli deterministický konečný automat. Pro tento účel byly využity webové formuláře, které ovšem jsou dynamicky rozšiřitelné, aby uživateli umožnily zadat opravdu jakýkoli počet stavů a vstupních symbolů automatu. Pro zadání přechodové funkce bylo vhodné, aby aplikace byla schopna vygenerovat tyto tabulky přechodů v podobě formuláře, jen na místě přechodů by místo pevné hodnoty měl být objekt select, ve kterém již budou vyplněny všechny stavy. Uživatel poté pouze vybere jeden z těchto stavů. Tímto způsobem by měl být minimalizován počet operací, které uživatel musí provést.

Po vytvoření automatu by měl mít uživatel možnost si daný program nějakým způsobem uložit. Pro tento účel byl využit jazyk XML[3], do kterého je automat serializován a poté uložen do souboru, který si uživatel může uložit do svého počítače. Proto je také nutné, aby aplikace umožnila vytvořit automat pomocí tohoto souboru.

Dále je nutné, aby aplikace umožnila kdykoli se ke konkrétní definici automatu vrátit a případně ji upravit, ať už pro případ chyby, nebo testu jiné konfigurace automatu. Tento požadavek byl zajištěn pomocí prostředku session, do kterého se po

[3] eXtensible Markup Language

každé změně ukládá aktuální serializovaná konfigurace automatu, ze které je tedy možné zpětně vytvořit vyplněné formuláře s danou konfigurací.

3.4 NAVRHOVANÁ VYLEPŠENÍ

Aplikace vytvořená v rámci magisterského projektu splňovala základní požadavky, ovšem jevilo se jako vhodné přidat další funkčnost, která by zjednodušila a rozšířila možnosti použití. Navíc se časem objevilo několik chyb, které bylo vhodné odstranit. Například animace při určitém postupu nefungovala korektně. Cílem tedy bylo, pomocí vhodného testování nalézt co největší množství dalších chyb a ty následně opravit.

Při návrhu také původně nebylo počítáno s možností potřeby rychlého přepínání několika automatů. Tento problém sice částečně řešila možnost zadání automatu pomocí XML souboru, ovšem tento způsob není příliš komfortní. Proto byla do aplikace přidána možnost uložit automat pomocí localStorage do webového prohlížeče, odkud je možné jej jedním kliknutím vložit do aplikace. Dalším vhodným vylepšením aplikace se jevilo přidání možnosti vyhodnocení většího množství slov najednou. Uživatel tak například může vložit do aplikace soubor se slovy a aplikace vyhodnotí, zdali je konkrétní slovo rozpoznatelné definovaným automatem nebo není, případně jestli nastala nějaká chyba při vyhodnocování. Výsledek je poté zobrazen v přehledné tabulce. Poslední navrhovanou změnou bylo vylepšení vzhledu aplikace, které je pouze kosmetickou úpravou, avšak uživateli velmi vnímanou.

Obrázek 3.3 – Zjednodušený vývojový diagram aplikace

3.5 TESTOVÁNÍ BĚHEM VÝVOJE

Vzhledem k typu aplikace je zřejmé, že nejdůležitějšími externími hledisky pro testování aplikace jsou spolehlivost a použitelnost. Proto byl během vývoje kladen důraz na funkční testování a testování použitelnosti. Pro funkční testování byla využita aplikace Selenium IDE.

Dále bylo rozhodnuto, že během vývoje bude použito jednotkové testování, zejména kvůli znovupoužitelnosti kódu. Nestandardní situace, kdy bylo navázáno na již do jisté míry vytvořenou aplikaci, byla vyřešena tím, že testy k již hotovým částem kódu byly zpětně vytvořeny. Poté se pokračovalo ve vývoji další funkcionality. Tento přístup sice není podle vývojových metodik zcela správný, ovšem výsledkem dodatečného vytvoření jednotkových testů byl potvrzen přínos jednotkového testování. Neboť byl testován kód, který již v rámci základní funkčnosti aplikace otestován byl.

Pro ladění vzhledu aplikace byl využit plugin Firebug, který výrazným způsobem urychlil vývoj.

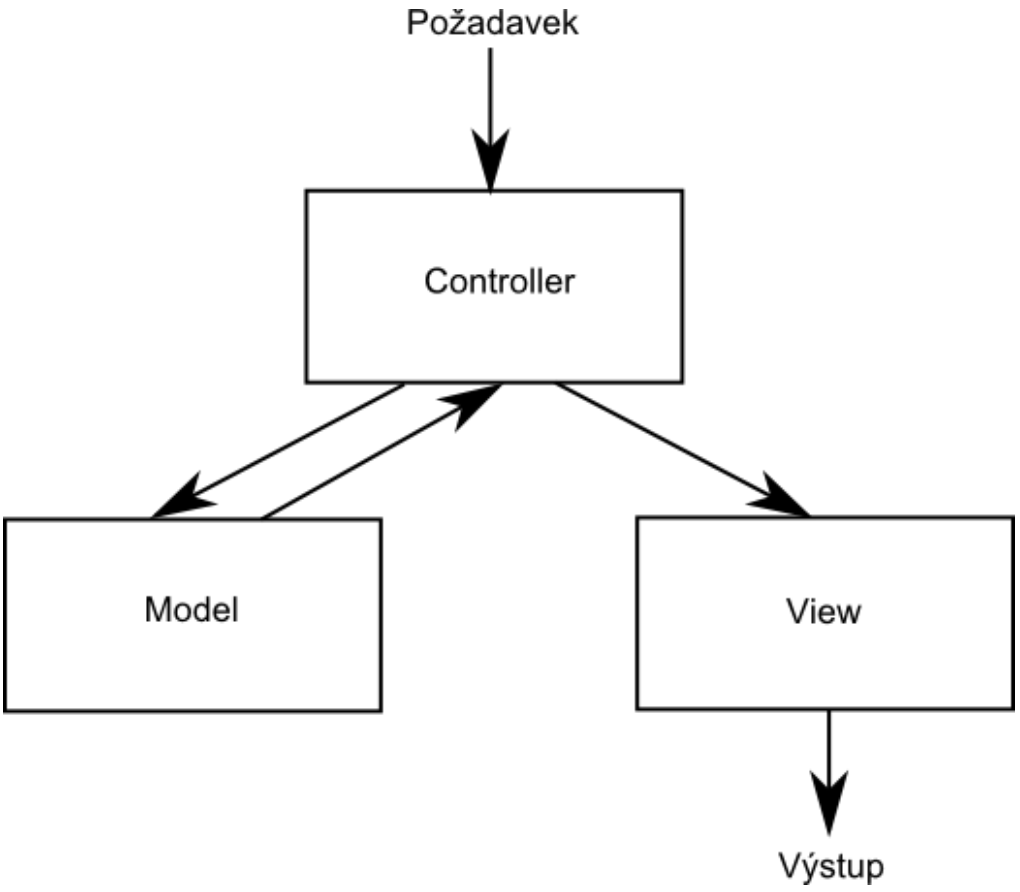
Po vytvoření aplikace a základním ověření funkčnosti bylo provedeno testování použitelnosti na určitém vzorku uživatelů. Cílem bylo nalezení co největší množství nedostatků, které byly posléze odstraněny.

Dále byl také kladen důraz na další testovací přístupy, jejichž použití bylo zváženo během vývoje. Cílem bylo použít co největšího množství testů, ovšem s hlavním ohledem na kvalitu aplikace. Výsledkem by měla být co nejkvalitnější, rozšiřitelná a použitelná aplikace.

4 PRAKTICKÁ ČÁST

4.1 IMPLEMENTACE APLIKACE

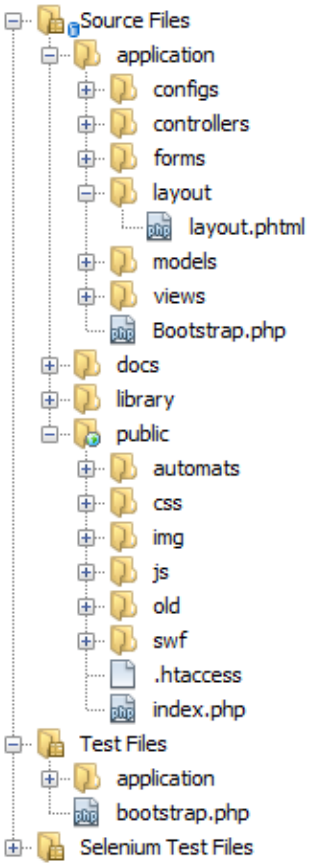
Dle návrhu byla postavena na PHP frameworku Zend, který používá architekturu MVC. Při realizaci byl kladen patřičný důraz na správné strukturování kódu. Hlavní operace s daty se tedy provádí v třídách typu model. Generování obsahu webových stránek se provádí v třídách typu view a zpracování požadavků probíhá v třídách typu controller. Architektura MVC je znázorněna na obrázku 4.1. Během vývoje byly také v omezení míře použity zdroje [20] až [23].



Obrázek 4.1 – Schéma architektury Model-View-Controller

4.1.1 STRUKTUROVÁNÍ PROJEKTU

Vzhledem existenci předpokladu, že aplikace bude dále rozšiřována, bylo vhodné aplikaci rozumně strukturovat. Základní struktura projektu je znázorněna na obrázku 4.2.

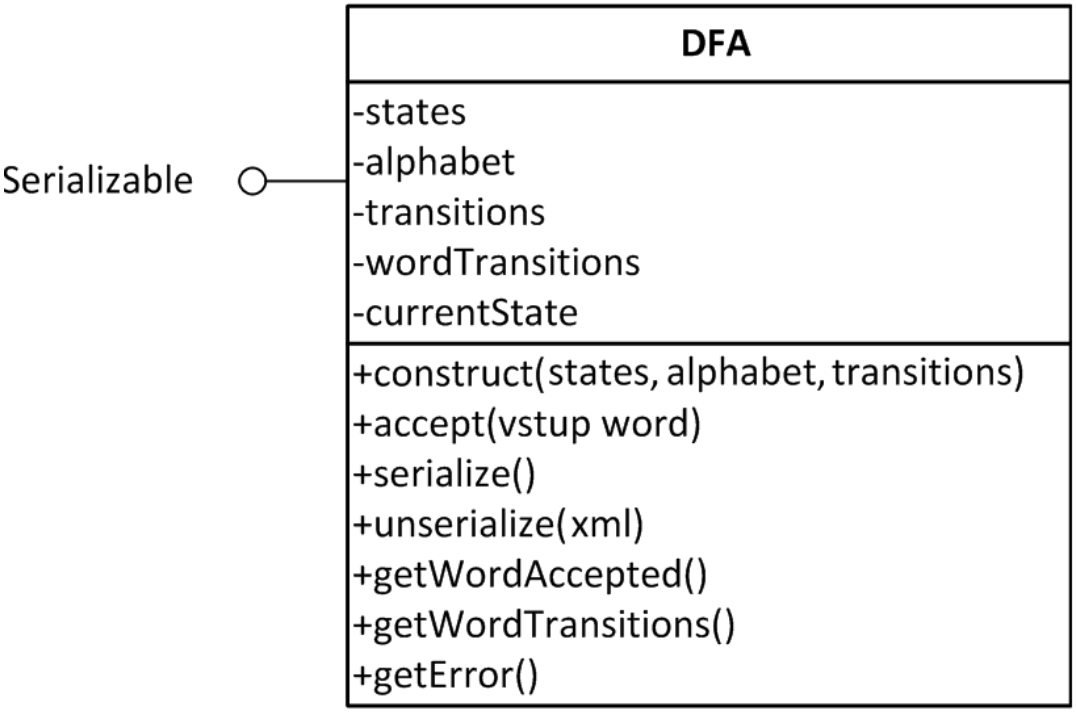


Obrázek 4.2 – Základní struktura projektu

Do složky public jsou uloženy všechny soubory, které budou přístupné z internetu. Ve složce application se nacházejí controllery, modely, pohledy a formuláře, které aplikace obsahuje. Tato složka by neměla být v produkčním stavu viditelná z internetu zejména kvůli bezpečnosti. Do složky docs se generuje dokumentace. Složka library obsahuje Zend Framework a další knihovní třídy. Ve zbylých složkách se nacházejí testy k aplikaci, ať už jednotkové nebo automatické.

4.1.2 KONEČNÝ AUTOMAT

Základním úkolem bylo vytvořit model, který by simuloval činnost deterministického konečného automatu. Třída pro tento úkol byla pojmenována zkratkou DFA, která vychází anglického překladu termínu deterministický konečný automat – deterministic finite automata. Činnost tohoto modelu je dána dle již zmíněné definice. Model si ve vnitřních proměnných states, alphabet a transitions udržuje svou konfiguraci. Také si udržuje aktuální stav, ve kterém se nachází. Nejdůležitější metodou je metoda accept, která slouží pro vložení slova na vstup automatu. Po vložení slova na vstup automatu proběhne jeho vyhodnocení. Model dále poskytuje metody pro zjištění, zdali bylo slovo automatem rozpoznáno, či nikoliv a dále v případě chyby při rozpoznávání slova vrací chybové hlášení, kde konkrétně při průchodu nastala chyba. Vhledem k tomu, že cílem této aplikace je také animovat tento průchod, poskytuje model metodu getWordTransitions, která vrací konkrétní posloupnost stavů, ve kterých se automat při průchodu slova nacházel.



Obrázek 4.3 – Zjednodušený UML diagram modelu konečného automatu

Jak již bylo zmíněno, pro ukládání modelu automatu byl zvolen jazyk XML, proto model automatu implementuje rozhraní Serializable. Model dále obsahuje několik dalších vnitřních proměnných a metod, které byly implementovány s ohledem na možné budoucí použití. Model si například uchovává, jaké bylo poslední rozpoznávané slovo a zdali bylo rozpoznáno. Tyto funkce nebyly ve vyvíjené aplikaci potřeba, ovšem při návrhu a implementaci modelu se zdálo rozumné je do modelu zahrnout. Zjednodušený UML diagram třídy DFA je znázorněn na obrázku 4.3.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DFA>
 <alphabet>
  <char>0</char>
  <char>1</char>
 </alphabet>
 <states>
  <state>
   <name>q1</name>
   <status>start</status>
  </state>
  <state>
   <name>q2</name>
   <status>final</status>
  </state>
 </states>
 <transitions>
  <transition>
   <state>q1</state>
   <onChar>
    <char>0</char>
    <state>q1</state>
   </onChar>
   <onChar>
    <char>1</char>
    <state>q2</state>
   </onChar>
  </transition>
  ...
 </transitions>
</DFA>
```

Obrázek 4.4 – Serializovaná podoba modelu automatu

4.1.3 ZADÁNÍ A ÚPRAVY KONEČNÉHO AUTOMATU

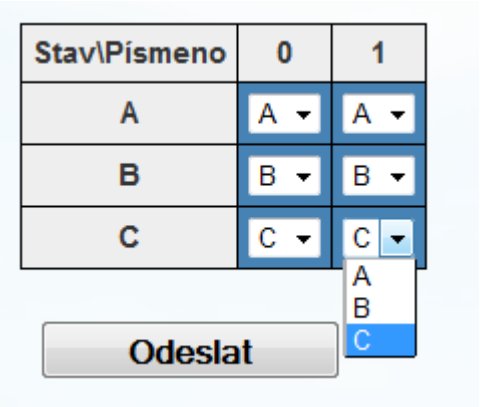
Aplikace nabízí uživateli tři možnosti, jak automat zadat. Pomocí formulářů, vložením XML souboru nebo načtením z localStorage. Při návrhu a implementaci

rozhraní pro zadání automatu pomocí formulářů byl kladen hlavní důraz na co možná nejvyšší jednoduchost tohoto postupu. Jako nejlepší se v této situaci jevilo použít dynamicky generované a upravované formuláře.

Nejprve je potřeba zadat množinu stavů automatu a abecedu automatu. Tento úkon je realizován dynamicky upravovatelnými formuláři. Dynamické úpravy formulářů jsou v tomto případě realizovány pomocí jazyka JavaScript. Uživateli je umožněno přidávat a odebírat řádky v těchto formulářích, jak je znázorněno na obrázku 4.5. Před odesláním formuláře je prováděna validace všech položek formuláře, zdali nejsou prázdné. Pokud je některá z nich prázdná, odeslání formuláře se neprovede a aplikace vygeneruje chybovou hlášku.

Obrázek 4.5 – Formulář pro zadání stavů a abecedy

Na základě zadaných stavů a abecedy automatu aplikace vygeneruje formulář pro zadání přechodové funkce automatu. Standardně je přechodová funkce nastavena tak, že automat zůstává pokaždé ve stejném stavu. Jak je znázorněno na obrázku 4.6. Po odeslání tohoto formuláře je již konečný automat vytvořený.



Obrázek 4.6 – Formulář pro zadání přechodové funkce

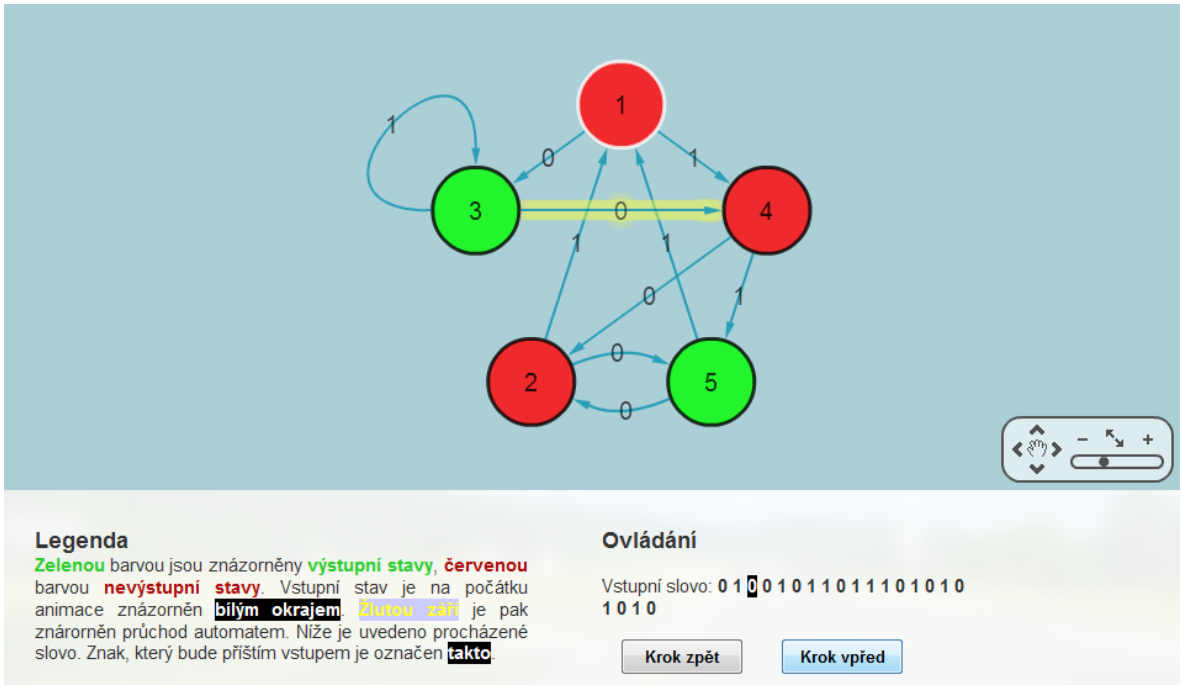
Při návrhu bylo také třeba zohlednit možnost pozdější úpravy automatu, například z důvodu chybného zadání nebo testovaní jiné konfigurace. Aplikace si udržuje aktuální konfiguraci automatu pomocí proměnných sessions, která jsou uložena na serveru a umožňují předávání dat mezi jednotlivými stránkami aktuální relace. To znamená, že veškeré změny v konfiguraci automatu jsou ihned ukládány do proměnných sessions a naopak aktivní stránka se na začátku vždy nejprve pokusí načíst data ze sessions, aby na jejich základě příslušně upravila své dynamické prvky. V případě, že pokus o načtení dat ze sessions není úspěšný, je stránka vygenerována na základě standardního nastavení. Uživatel je tedy poté schopen se prakticky kdykoli vrátit k definici automatu pomocí zpětných odkazů v aplikaci. Odkazy zpět fungují jako inverzní postup zadání automatu pomocí formulářů, tedy pokud je automat zadán a uživatel požaduje krok zpět, je přesměrován na stránku s konfigurací přechodové funkce, kterou může upravit, nebo případným dalším krokem zpět přesměruje aplikaci na stránku pro zadání množiny stavů a abecedy automatu. Je třeba zmínit, že při zpětném průchodu konfigurace automatu jsou formuláře generovány do podoby aktuální konfigurace automatu, což usnadňuje úpravy a zlepšuje uživateli orientaci.

4.1.4 ANIMACE

Animace byla vytvořena pomocí knihovny Cytoscape Web. Dle návrhu aplikace funguje tak, že na vstup vytvořeného automatu je vloženo slovo pro zpracování a

výsledek tohoto zpracování je předán dále klientovi, jehož úkolem je již samostatně zabezpečit animaci průchodu slova automatem. Klient tedy od serveru obdrží informace o jednotlivých stavech automatu, zpracovávané slovo a pořadí stavů, v jakém se bude automat postupně při průchodu slova nalézat.

Animace průchodu slova je zajištěna pomocí grafu přechodů. Oproti standardním zvyklostem bylo zvoleno jiné označení vstupních a výstupních stavů. Místo používaných šipek bylo zvoleno barevné znázornění zejména kvůli názornosti. Výstupní stavy jsou obarveny zelenou barvou a nevýstupní stavy jsou obarveny červenou barvou. Pokud tedy průchod slova skončí v zeleně obarveném stavu, slovo bylo rozpoznáno a pokud skončí v červeně obarveném stavu, slovo rozpoznáno nebylo. Vstupní stav je označen bílou barvou okraje, zatímco ostatní stavy mají okraj černý. Vstupní stav je také zdůrazněn na začátku animace žlutým stínem, který znázorňuje, v jaké části grafu se aktuálně automat nachází. Ukázka animace je uvedena na obrázku 4.7.

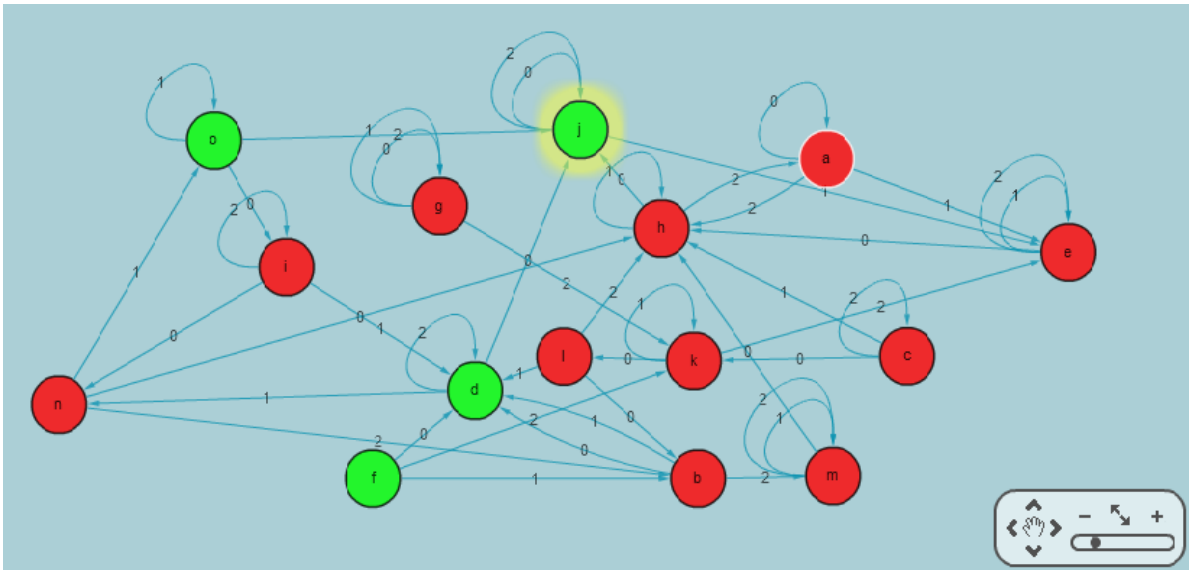


Obrázek 4.7 – Ukázka animace průchodu slova automatem

Interakce s uživatelem je zajištěna pomocí tlačítek, jimiž lze postupně procházet zadané slovo po jednotlivých znacích. Na tyto tlačítka jsou navázány JavaScriptové funkce, které ovládají grafické znázornění. Vzhledem k tomu, že je animace zajištěna

pouze pomocí lokálního klienta, není závislá na rychlosti sítě a odezva na prováděné operace je tedy velmi rychlá.

Aby byla zajištěna co největší míra interaktivnosti animace, je možné polohu stavů uživatelsky upravovat. Tato operace je zajištěna pomocí známé funkce drag and drop. Uživatel tedy může pomocí myši jakýkoli stav uchopit a umístit jej do polohy, která mu bude lépe vyhovovat. Dále je také možné dynamicky měnit velikost znázorňovaného grafu, což je vhodné v situaci, kdy je animován automat s větším množstvím stavů, protože se může velmi jednoduše stát, že při větším množstvím stavů, bude graf nepřehledný. Uživatel tedy může upravovat kromě polohy jednotlivých stavů také celkovou velikost znázorňovaného grafu. Příklad grafu, kde se použití zmíněných úprav jeví jako nezbytné, je uveden na obrázku 4.8.

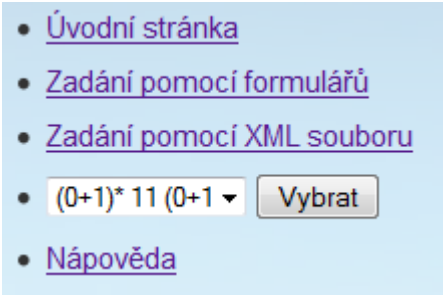


Obrázek 4.8 – Ukázka složitějšího grafu přechodů

4.1.5 LocalStorage

LocalStorage je úložiště, které poskytují moderní webové prohlížeče. V této aplikaci bylo použito pro ukládání serializovaných konfigurací automatů. Pokud tedy webový prohlížeč localStorage poskytuje, může do něj uživatel po vytvoření automatu tuto konfiguraci uložit. Automat je ukládán pod textovým jménem zadaným uživatelem. Lze použít například regulární výraz.

Aplikace po každém obnovení testuje, zdali je localStorage povoleno a případně, zdali obsahuje uložené automaty. Pokud ano, přidá do menu aplikace položku pomocí níž lze vytvořit automat načtení právě z localStorage. Ukázka menu s položkou localStorage je znázorněna na obrázku 4.9. Pokud localStorage prohlížeč nepodporuje, nebo úložiště žádný automat neobsahuje, není tato položka v menu zobrazena.



Obrázek 4.9 – Ukázka menu s aktivním localStorage

LocalStorage bylo pro ukládání automatů zvoleno zejména kvůli jednoduchosti ukládání. Pokud by totiž bylo zvoleno pro ukládání automatů centrální úložiště, bylo by potřeba řešit přiřazení automatů k jednotlivým uživatelům, což by pro uživatele znamenalo nutnost registrace. Tento přístup se nejevil jako nejefektivnější, protože bylo předpokládáno, že kvůli možnosti uložení automatu by se registrovalo jen mizivé procento uživatelů a funkce by tak nebyla příliš využita. Použití localStorage je sice omezeno pouze na konkrétní počítač a prohlížeč, ovšem pokud vezmeme v úvahu, že hlavním požadavkem bylo poskytnout uživateli možnost jednoduše a rychle měnit konfiguraci testovaného automatu, jeví se požití localStorage jako rozumné. Tím spíše, když aplikace umožňuje uživateli konfiguraci automatu uložit v podobě serializovaného XML souboru, což řeší právě problém s přenositelností automatu mezi jednotlivými počítači a prohlížeči.

4.1.6 ZPRACOVÁNÍ SOUBORU S VÍCE SLOVY

Aplikace dále umožňuje vložit soubor se slovy, která má daný automat zpracovat. Formát souboru se slovy byl volen s ohledem na maximální jednoduchost tak, aby jej uživatel mohl co nejjednodušeji a co možná nejrychleji vytvořit. Proto byl zvolen formát, kdy jeden řádek souboru reprezentuje jedno slovo,

které se má zpracovat. Slova jsou tedy načtena zpracováním jednotlivých řádků vloženého souboru. Po načtení jsou postupně vkládána na vstup zadaného automatu a poté je zkoumáno, jak rozpoznávání dopadlo. Po zpracování všech slov automatem je vygenerována tabulka s výsledky, která je prezentována uživateli. Příklad této tabulky je znázorněn na obrázku 4.10.

Během testování této části se ukázalo, že tato funkce je použitelná řádově pro tisíce slov o průměrné délce 15 znaků. Bylo by tedy možné soubory se slovy generovat programově například na základě zvoleného jazyka a pomocí této funkce ověřit, zdali automat tento jazyk rozpoznává, či nikoliv. Čímž se z této aplikace stává mnohem použitelnější nástroj, než bylo původně zamýšleno.

Obrázek 4.10 – Zpracování více slov

4.2 JEDNOTKOVÉ TESTOVÁNÍ

Při vývoji aplikace bylo použito jednotkové testování pomocí testovacího frameworku PHPUnit. Jak již bylo zmíněno v úvodu, aplikace byla do jisté míry rozpracovaná během magisterského projektu, proto bylo potřeba nejdříve upravit stávající kód aplikace, aby na něj mohly být použity jednotkové testy a poté dopsat jednotkové testy k již hotovým částem aplikace. Tímto přístupem se naskytlo zajímavé srovnání, kdy byl již funkčně otestovaný kód testován znovu. V již otestovaném kódu bylo nalezeno jen minimální množství chyb. Ovšem v aplikaci bylo také množství kódu, který otestován nebyl, jako například chybová hlášení nebo některé metody, které

v aplikaci nebyly použity, ale v kontextu svých tříd mají opodstatnění. V těchto metodách bylo nalezeno relativně velké množství chyb.

Po dopsání jednotkových testů a úpravě stávajícího kódu aplikace tak, aby testy proběhly úspěšně, byla aplikace dále vyvíjena podle standardní metodiky TDD[4], tedy nejprve byl vytvořen test, a až poté byl vytvořen výkonný kód metody. Vzhledem k tomu, že integrované části již byly otestovány, probíhala integrace mnohem rychleji, protože nebylo potřeba tolik času k ladění. Jednotkové testy se osvědčily hlavně u tříd typu model, ve kterých dochází ke zpracování dat. Tyto třídy tvoří hlavní jádro aplikace, a proto je velmi vhodné je pečlivě otestovat. V případě vytvářené aplikace se neukázalo jako příliš přínosné testovat třídy typu controller, neboť v aplikaci jsou využity především pro předání dat třídám typu view a jejich funkčnost lze mnohem snadněji otestovat uživatelským testováním. Ovšem lze si představit případy, kdy může controller zabezpečovat složitější operace a zde by tedy jednotkové testování zcela jistě smysl mělo. Také u tříd typu view se nejevilo jako vhodné použít jednotkové testování, neboť v projektu byly použity pouze jako šablony pro generování výstupního HTML kódu stránky.

Při vývoji bylo také vyzkoušeno jednotkové testování JavaScriptu. Konkrétně aplikací FireUnit. V aplikaci byl JavaScript použit hlavně jako doplněk pro dynamické chování stránek. Většina funkcí byla navázána k nějakým objektům, například k tlačítkům. Bohužel se jednotkové testování v tomto případě ukázalo jako nepříliš použitelné, neboť aplikace FireUnit sice umožňuje programově generovat uživatelské akce, jako například vyplnění textového pole nebo stisk tlačítka, ovšem již dále nedochází ke spuštění funkcí, které jsou k těmto událostem připojeny, proto je nebylo možné tímto způsobem otestovat. V tomto případě se potvrdilo, že jednotkové testování se hodí hlavně na testování funkcí, tříd nebo malých částí kódu ve smyslu testovaní návratových hodnot, nikoli k testování uživatelské interakce. Vzhledem k tomu, že byl JavaScript použit hlavně pro dynamickou úpravu stránek, nemělo jednotkové testování JavaScriptu valný význam. Navíc je toto testování relativně pracné, neboť je třeba pro tyto testy vytvořit speciální stránku, která je poté zpracována testovací aplikací.

[4] Test driven development – vývoj řízený testováním

```
protected function setUp() {
    $states = array("Q1" => "start", "Q2" => "none", "Q3" =>"final");
    $alphabet = array(1=>"0", 2=>"1");
    $transitions["Q1"]["0"]="Q1";
    $transitions["Q1"]["1"]="Q2";
    $transitions["Q2"]["0"]="Q2";
    $transitions["Q2"]["1"]="Q3";
    $transitions["Q3"]["0"]="Q3";
    $transitions["Q3"]["1"]="Q3";
    $this->object = new Application_Model_DFA($states, $alphabet,
            $transitions);
}

public function testGetWordTransitions() {
    $this->object->accept("00000111111");
    $this->assertEquals(array( 0=> "Q1", 1=>"Q1", 2=> "Q1", 3=>"Q1",
        4=> "Q1", 5=>"Q1", 6=>"Q2",7=> "Q3", 8=> "Q3", 9=>"Q3",
        10 =>"Q3", 11 => "Q3"),
        $this->object->getWordTransitions());
}
```

Obrázek 4.11 – Ukázka jednotkového testu modelu DFA

Jednotkové testování má zcela jistě pozitivní účinky na údržbu a znovupoužití kódu. Zároveň slouží jako velmi kvalitní dokumentace. V případě slabě typovaného jazyka PHP to platí dvojnásob. Na druhou stranu náklady potřebné na vytvoření jednotkových testů jsou nemalé. Odhadem by bylo možné říci, že se čas potřebný na vývoj zdvojnásobí. Zároveň je potřeba tvorbu kódu jistým způsobem přehodnotit, tak aby bylo možné jej testovat. Na základě informací z literatury a zkušeností získaných při tvorbě této aplikace je možné dojít k závěru, že jednotkové testování je přínosné v případě, že je prováděno důsledně a správně. Důsledně ve smyslu testování každé jednotlivé komponenty, funkce nebo metody a správně ve smyslu, že testy opravdu ověřují funkčnost testovaného kódu. Při splnění těchto podmínek se vytvářený kód stává robustnějším a obrovským způsobem se zvyšuje jeho možná znovupoužitelnost. Nárůst nákladů při tvorbě kódu by měl být kompenzován nižšími náklady při pozdějších úpravách a znovupoužití kódu.

4.3 AUTOMATIZOVANÉ TESTOVÁNÍ

Pro automatizované testování byl použit program Selenium IDE. Hlavním přínosem programu je možnost zaznamenat uživatelskou činnost, vytvořit validační

pravidla a test poté automaticky spouštět. Ukázalo se, že tento přístup má obrovský přínos zejména z hlediska úspory časových nákladů při ladění aplikace. Například při ladění vytváření automatu bylo potřeba vyplnit dva formuláře, ve kterých se definovala konkrétní konfigurace automatu. Pokud bychom k dispozici automatizované testování neměli, znamenalo by to, že pro každý test je potřeba tyto formuláře vyplnit znovu. Pokud vezmeme v úvahu fakt, že při ladění není možné krokovat zpět, znamenalo by to, že pro každý jednotlivý ladící průchod je potřeba ručně tyto formuláře vyplnit. Což je při potřebě většího množství ladících průchodů časově i psychicky náročné. Použitím programu Selenium IDE spolu s ladícím nástrojem XDebug se ladění zefektivnilo v maximální možné míře. Uživatelskou činnost stačilo provést pouze jednou, přičemž byla zaznamenána Seleniem, a pro každé další použití již byla použita automatizovaná podoba.

Automatizované testování je možné také využít při uživatelském testování podle scénářů. Pokud bude tester svou práci zaznamenávat pomocí Selenia, získáme sadu scénářů, které je poté možné spouštět později. Také při hlášení chyb může tento přístup přinést zefektivnění. Tester musí po nalezení nějaké chyby poskytnout vývojovému týmu co největší množství informací o chybě, podmínkách a akci, která tuto chybu způsobila. Pokud tuto akci zaznamená pomocí Selenia, slouží tento automatický scénář jako dokonalá informace o tom, jak byla chyba nalezena. Vzhledem k tomu, že testovací prostředí umožňuje test procházet po jednotlivých krocích, je velmi snadné, aby si programátor scénář prošel a pochopil, kde se chyba vyskytuje. Samozřejmě je vhodné tento scénář doplnit o další informace.

V teoretické části bylo zmíněno, že Selenium umožňuje pomocí části WebDriver vytvářet testy programováním. Za cenu vyšších nákladů na tvorbu testu umožňuje tento přístup testovat aplikaci komplexněji a na velmi nízké úrovni. Použití tohoto přístupu se ovšem jevilo jako příliš nízkoúrovňové pro testování řešené aplikace.

4.4 TESTOVÁNÍ POUŽITELNOSTI

Uživatelské rozhraní aplikace bylo otestováno testem použitelnosti, který byl realizován na vzorku tří uživatelů znalých problematiky konečných automatů. S každým uživatelem byl realizován tentýž uživatelský scénář, který byl vytvořen tak, aby otestoval co největší rozsah aplikace.

Ukázka z testovacího scénáře

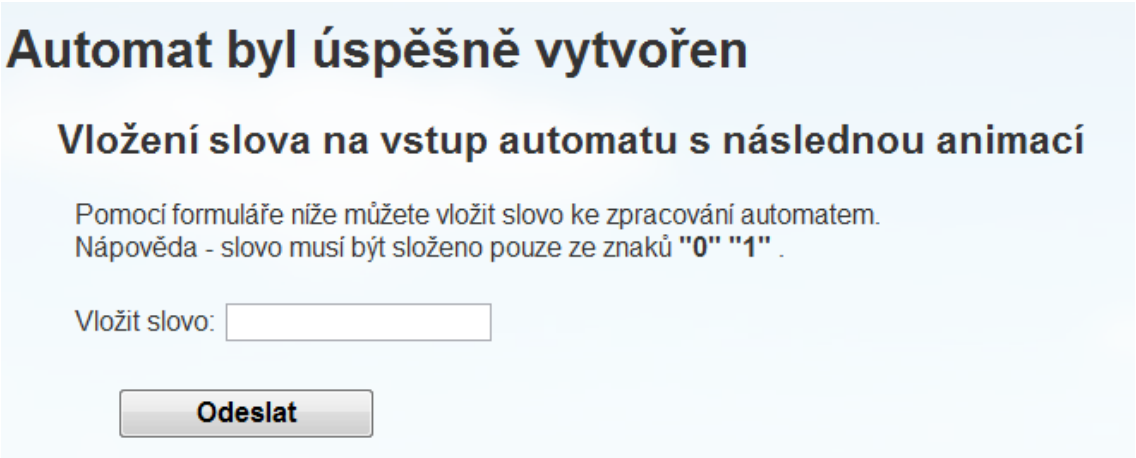
1. Spusťte svůj oblíbený prohlížeč a jděte na stánku http://localhost/diplomka/public/.
2. Vytvořte automat se třemi stavy 1, 2, 3 a dvěma vstupními symboly A a B. První stav označte jako vstupní, poslední jako výstupní.
3. Přechodovou funkci vytvořte tak, že na znak A automat vstoupí do vyššího stavu, pokud vyšší stav není, automat zůstane ve stávajícím stavu. Stejně tak na znak B automat zůstane v současném stavu.
4. Vytvořený automat si uložte na disk v podobě XML souboru.
5. Vložte slovo AAAAABABABA.
6. Upravte si polohu stavů automatu tak, aby vám vyhovovala.
7. Projděte slovo automatem.
8. Bylo slovo automatem rozpoznáno?
9. Vložte další slovo AA.
10. Projděte automatem slovo a zjistěte, zdali bylo rozpoznané.
11. Jděte zpět na úvodní stránku.
12. Vytvořte automat pomocí uloženého XML souboru.
13. …

Pokud to bylo možné, bylo řešení zadaných úkolů ponecháno pouze na uživateli. Rada byla uživateli poskytnuta pouze v případě, že si uživatel vůbec nevěděl se zadaným úkolem rady. Během testování byla velmi pečlivě sledována činnost uživatelů a komunikován jejich pocit z ovládání. Po provedení testu byli uživatelé požádáni, aby vyjádřili svůj názor na aplikaci a případně sdělili své připomínky.

Po otestování uživatelského rozhraní na zmíněném vzorku uživatelů bylo potřeba vyhodnotit výsledky tohoto testu. Zde je potřeba říci, že uživatelé se ve většině připomínek shodovali. Nejdůležitější připomínkou bylo, že k aplikaci by bylo vhodné doplnit nápovědu, kde bude vysvětleno jak aplikaci používat. Dalším objeveným nedostatkem bylo ukládání XML souboru. Po kliknutí na odkaz pro stažení souboru docházelo k otevření XML souboru přímo v prohlížeči, což nebylo v tomto případě vhodné. Ideální by bylo, kdyby po kliknutí na odkaz ihned došlo ke stažení souboru na uživatelský počítač. Poslední připomínkou bylo, že na stránce pro zadávání stavů

automatu a vstupních symbolů by bylo vhodné prohodit pořadí formulářů a tedy, že by první měl být formulář pro zadání stavů a dále formulář pro zadání vstupních symbolů. Pozorováním činnosti uživatelů bylo také zjištěno, že by bylo vhodné do určitých částí aplikace přidat doprovodný popisný text s případnou nápovědou. Například při vkládání slova pro zpracování automatem by bylo vhodné uživateli napovědět, z jakých symbolů by se slovo mělo skládat.

Nalezené chyby byly dle možností opraveny. K aplikaci byla přidána nápověda. Stažení automatu bylo upraveno tak, aby opravdu po kliknutí na odkaz došlo ke stažení automatu v podobě XML souboru. Pořadí formulářů při zadávání stavů automatu a vstupních symbolů bylo upraveno. A do aplikace byly přidány popisky, které by měly uživateli pomoci s ovládáním aplikace. Jednu z těchto úprav je možné vidět na obrázku 4.12.



Obrázek 4.12 – Upravené rozhraní aplikace

4.5 VALIDACE A TESTOVÁNÍ V RŮZNÝCH PROHLÍŽEČÍCH

Během vývoje byla aplikace průběžně testována vůči HTML a CSS validátorům. Z tohoto hlediska se neobjevil žádný zásadní problém. Nalezené chyby byly ve většině případů způsobeny překlepy a jen v minimu případů bylo potřeba prostudovat specifikaci a kód náležitým způsobem upravit. Několik stránek aplikace obsahuje drobné chyby z hlediska HTML kódu, ovšem tyto chyby jsou způsobeny kódem generovaným pomocí Zend frameworku. Vzhledem k tomu, že se jednalo pouze

o opravdu drobné chyby, bylo rozhodnuto, že se opravovat nebudou, protože jejich oprava by mohla být časově náročná, ovšem s nulovým efektem ve smyslu použitelnosti, nebo funkčnosti aplikace.

Funkčnost aplikace byla také otestována v různých webových prohlížečích. Konkrétně v prohlížečích Internet Explorer 9, Google Chrome 18, Mozilla Firefox 12 a Opera 11. Aplikace je v těchto prohlížečích funkční ve všech svých aspektech. Důležité bylo v tomto ohledu zejména ověření funkčnosti animace a úložiště localStorage. Grafická interpretace aplikace se v jednotlivých prohlížečích v některých minimálních detailech liší, což je způsobeno již zmíněnou rozdílnou implementací jednotlivých prohlížečů. Optimalizace vzhledu aplikace vzhledem k různým prohlížečům byla provedena jen okrajově s cílem, aby aplikace byla v různých prohlížečích především použitelná. Usilovat o zcela identický vzhled aplikace ve všech prohlížečích se s ohledem na druh a použití aplikace jevilo jako neúčelné.

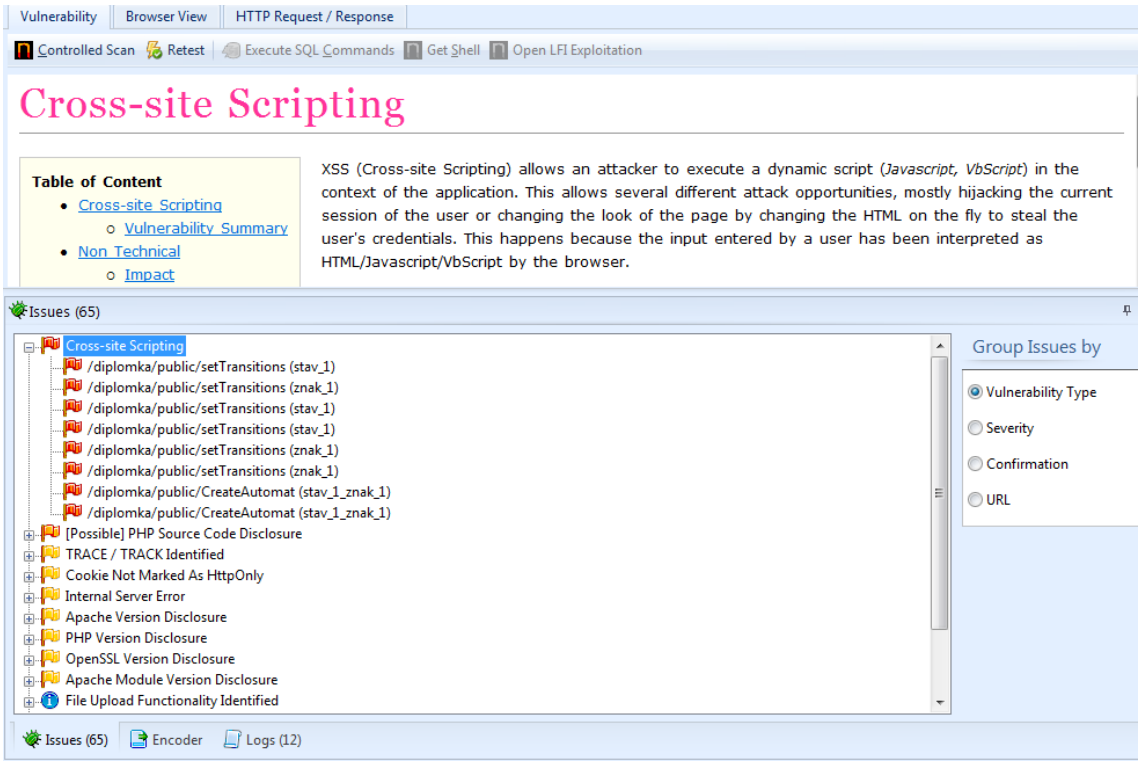
4.6 BEZPEČNOSTNÍ TESTOVÁNÍ

Testování z bezpečnostního hlediska bylo provedeno programem Netsparker. Byly nalezeny dvě závažnější chyby. První nalezenou chybou byla náchylnost aplikace k XSS[5] útoku, což je útok, při kterém je do aplikace vložen cizí programový kód pomocí neošetřených vstupů formulářů. Útočník tímto způsobem může měnit vzhled stránky nebo upravovat její funkčnost. Řešením tohoto problému je ošetření vstupů aplikace z formulářů proti znakům < a >. Buďto jejich nahrazení HTML entitami, nebo jejich úplným vymazáním, neboť ve většině případů nemají pro aplikaci opodstatnění. Řešení této chyby bylo zabezpečeno pomocí funkce htmlspecialchars, která přepisuje speciální znaky, které mají význam v HTML jazyce na HTML entity. Touto funkcí byl ošetřen každý textový vstup aplikace.

Druhá nalezená bezpečnostní chyba byla nalezena v souboru layout.phtml, který obsahuje HTML kostru aplikace. Bezpečnostní riziko je v tom, že tento soubor obsahuje PHP kód, ke kterému by se mohl útočník dostat. V případě, že by tento soubor obsahoval citlivé informace, jako například připojení k databázi nebo jméno a heslo, byla by tato stránka opravdu velkým bezpečnostním rizikem. Ovšem v našem případě v tomto souboru dochází pomocí PHP pouze k dynamickému generování odkazů na

[5] Cross-site scripting

podstránky aplikace. A pokud by se útočník k tomuto souboru dostal, zjistil by z něj pouze jména controllerů aplikace, což nepředstavuje žádné bezpečnostní riziko.



Obrázek 4.13 – Výsledky bezpečnostního testu v programu NetSparker

5 ZÁVĚR

V teoretické části diplomové práce bylo definováno několik základních pojmů a zpracována problematika testování webových aplikací. Pro jednotlivé typy testování byly vyzkoušeny a zhodnoceny v praxi nejvíce rozšířené nástroje. Výsledkem je ucelený přehled přístupů a nástrojů, kterými lze webové aplikace testovat. Vzhledem k současnému trendu přesunu většiny aplikací na internet se dá předpokládat, že se tato oblast bude dále velmi dynamicky rozvíjet. Neúplnost testovacích nástrojů je nejpatrnější u programovacího jazyka JavaScript, jehož použití se neustále posouvá.

V praktické části diplomové práce bylo navázáno na autorův magisterský projekt, ve kterém byl vyvíjen simulátor konečného automatu jako webová aplikace. Bylo navrhnuto několik dalších funkcí a úprav, například ukládání konfigurace jednotlivých automatů do localStorage. Během vývoje aplikace byla snaha použít co největší množství testovacích nástrojů s cílem vytvořit co možná nejkvalitnější aplikaci. Vytvořená aplikace umožňuje vytvořit model deterministického konečného automatu pomocí dynamických formulářů. Vytvořený automat je možné uložit do localStorage, nebo v podobě XML souboru. Vzhledem k tomu, že by aplikace měla sloužit jako pomůcka pro výuku, je nejdůležitější částí interaktivní animace znázorňující průchod slova konečným automatem. Animace je implementována tak, že je vykonávána pouze na straně klienta a není tak závislá na rychlosti internetového připojení. Aplikaci je také možné využit ke zpracování většího množství slov předaných aplikaci v podobě souboru.

Během vývoje bylo používáno jednotkové testování, které se osvědčilo zejména s ohledem na znovupoužitelnost kódu. Otestovaný kód je také více srozumitelný, neboť jednotkové testy poskytují velmi dobrou dokumentaci. Při vývoji bylo také využíváno automatizované testování přínosné zejména při ladění kódu. Použitím automatického testu se časové nároky na test snížily na minimum a zároveň testování probíhalo efektivněji.

Uživatelské rozhraní aplikace bylo otestováno v ohledu použitelnosti na vzorku tří uživatelů. Nalezené nedostatky byly upraveny a připomínky uživatelů byly do aplikace zapracovány. Uživatelé u aplikace oceňovali jednoduchost zadání automatu a také animaci, kterou hodnotili jako srozumitelnou a velmi povedenou. Testování použitelnosti bylo nejzajímavější částí testování zejména kvůli získaným názorům a

připomínkám od uživatelů. Aplikace také byla průběžně validována vzhledem k HTML a CSS kódu. Z hlediska HTML kódu aplikace obsahuje několik drobných chyb, které jsou generovány použitým frameworkem. Z hlediska CSS kódu je aplikace validní. Aplikace je funkční ve všech nejpoužívanějších webových prohlížečích. Jmenovitě jsou to Internet Explorer, Google Chrome, Mozilla Firefox a Opera.

Bylo také provedeno bezpečností testování, kterým byla objevena pouze jedna závažnější chyba, konkrétně náchylnost aplikace k cross-site sripting útoku. Tato chyba byla odstraněna důsledným ošetřováním uživatelských vstupů aplikace.

Do vyvíjené aplikace byly implementovány všechny funkce, které byly v rámci této práce navrženy. Aplikace je plně funkční a dle dostupných reakcí uživatelů použitelná a zajímavá. Využité testovací přístupy přispěly k objevení a opravě nemalého množství chyb. Testování aplikace by bylo zcela jistě efektivnější, kdyby bylo prováděno uživatelem, který není autorem aplikace. Bohužel tato podmínka byla splněna pouze při testování použitelnosti. V dalších oblastech testování nebyl takovýto přístup s ohledem na časovou náročnost testování a formát práce možný. V současné době je aplikace online dostupná na adrese http://simulator.cekuj.net. Vzhledem k tomu, že se jedná o bezplatný hosting, není možné garantovat její permanentní dostupnost.

SEZNAM POUŽITÉ LITERATURY

[1] DEVEL.CZ LAB. *Zdroják*: *o tvorbě webových stránek a aplikací* [online]. 2012 [cit. 2012-05-13]. ISSN 1803-5620. Dostupné z: http://www.zdrojak.cz/.

[2] CHYTIL, Michal. *Automaty a gramatiky*. Praha: SNTL - Nakladatelství technické literatury, 1984. matematický seminář SNTL.

[3] MCCONNELL, Steve. *Dokonalý kód*: *Umění programování a techniky tvorby software*. Dotisk prvního vydání. Brno: Computer Press, 2005, 894 s. ISBN 80-251-0849-X.

[4] PATTON, Ron. *Testování softwaru*: *automatické i ruční testování : testování použitelnosti, lokalizace i kompatibility produktů*. Brno: Computer Press, 2002. Pro každého uživatele. ISBN 80-7226-636-5.

[5] LECKY-THOMPSON, Ed, Steven D. NOWICKI a Thomas MYER. *Professional PHP6*. Indianapolis, Indiana: Wiley Publishing, Inc., 2009. Wrox Programmer to Programmer. ISBN 978-0-470-39509-7.

[6] BERGMANN, Sebastian. *PHPUnit Manual* [online]. 2005 [cit. 2012-04-23]. Dostupné z: http://www.phpunit.de/manual/3.7/en/index.html.

[7] FENTON, Steve. *Home - Enhance PHP Unit Testing And Mocking Framework : improve the quality* [online]. 2011 [cit. 2012-02-03]. Dostupné z: http://www.enhance-php.com/.

[8] *SimpleTest - Unit Testing for PHP* [online]. 2012 [cit. 2012-05-03]. Dostupné z: http://simpletest.org/.

[9] MOZILLA. *Firebug* [online]. 2012 [cit. 2012-05-03]. Dostupné z: http://getfirebug.com/.

[10] GIFT, Noah. *Functional testing for Web applications* [online]. 10.4.2009 [cit. 2012-04-02]. Dostupné z: https://www.ibm.com/developerworks/web/library/wa-aj-testing/.

[11] OPENQA. *Selenium - Web Browser Automation* [online]. 2012 [cit. 2012-05-03]. Dostupné z: http://seleniumhq.org/.

[12] *Windmill Testing Framework* [online]. 2007, 2010 [cit. 2012-05-16]. Dostupné z: http://www.getwindmill.com/.

[13] DOUŠA, Petr a Lukáš MARVAN. WebExpo 2010: Přednáška: Petr Douša, Lukáš Marvan - Testování použitelnosti – LIVE!. WEBEXPO. *Webexpo Prague Conference* [online]. 2010 [cit. 2012-05-03]. Dostupné z: http://webexpo.cz/praha2010/prednaska/testovani-pouzitelnosti-live/.

[14] KRUG, Steve. *Webdesign*: *Nenuťte uživatele přemýšlet*. 2. aktualizované vydání. Brno: Computer press, 2006. ISBN 80–251–1291–8.

[15] APACHE SOFTWARE FOUNDATION. *Apache JMeter* [online]. 2012 [cit. 2012-05-07]. Dostupné z: http://jmeter.apache.org/.

[16] MAVITUNA SECURITY. *Netsparker, False Positive Free Web Application Security Scanner* [online]. 2012 [cit. 2012-05-07]. Dostupné z: http://www.mavitunasecurity.com/netsparker/.

[17] ZEND TECHNOLOGIES LTD. *Zend Framework* [online]. 2006 [cit. 2012-04-12]. Dostupné z: http://framework.zend.com/.

[18] THE JQUERY FOUNDATION. *JQuery*: *The Write Less, Do More, JavaScript Library* [online]. 2012 [cit. 2012-04-12]. Dostupné z: http://jquery.com/.

[19] CYTOSCAPE CONSORTIUM. *Cytoscape Web* [online]. 2009, 27.3.2012 [cit. 2012-04-05]. Dostupné z: http://cytoscapeweb.cytoscape.org/.

[20] LUCCA, Giuseppe a Anna FASOLINO. Web Application Testing. MENDES, Emilia a Nile MOSLEY. *Web Engineering*. Berlin: Springer Berlin Heidelberg, 2006, s. 219-260. ISBN 978-3-540-28218-1.

[21] MALÝ, Martin. Ještě k testování - Zdroják. INTERNET INFO, s.r.o. *Zdroják - o tvorbě webových stránek a aplikací* [online]. 14.3.2011 [cit. 2012-04-10]. ISSN 1803-5620. Dostupné z: http://zdrojak.root.cz/clanky/jeste-k-testovani/.

[22] VRÁNA, Jakub. *1001 tipů a triků pro PHP*. Brno: Computer Press, 2010. ISBN 978-80-251-2940-1.

[23] MALÝ, Martin. Zombie, fantóm, bezhlavý rytíř... aneb Automatické zpracování webu - Zdroják. INTERNET INFO, s.r.o. *Zdroják - o tvorbě webových stránek a aplikací.* [online]. 17.6.2011 [cit. 2012-04-10]. ISSN 1803-5620. Dostupné z: http://zdrojak.root.cz/clanky/zombie-fantom-bezhlavy-rytir-aneb-automaticke-zpracovani-webu/

SEZNAM PŘÍLOH

Seznam příloh uložených na přiloženém médiu.

1. Text práce ve formátu PDF.
2. Text práce ve formátu docx.
3. Projekt aplikace s veškerými zdrojovými kódy a testy.
4. Knihovny Zend Framework a jQuery.